

Ion Lungu

BAZE DE DATE ORACLE
LIMBAJUL
SQL

EDITURA ASE

CAPITOLUL 1.

TEORIA BAZELOR DE DATE RELAȚIONALE

1.1. MODELUL RELAȚIONAL

Modelul relațional a fost *propus* de către IBM şi a revoluționat reprezentarea datelor făcând trecerea la generația a doua de baze de date.

Modelul este *simplu*, are o solidă *fundamentare teoretică* fiind bazat pe teoria seturilor (ansamblurilor) şi pe logica matematică. Pot fi reprezentate toate tipurile de structuri de date de mare complexitate, din *diferite domenii* de activitate.

Modelul relațional este definit prin: structura de date, operatorii care acționează asupra structurii şi restricțiile de integritate.

1) Conceptele utilizate pentru definirea **structurii de date** sunt: domeniul, tabela (relația), atributul, tuplul, cheia şi schema tabelei.

Domeniu este un ansamblu de valori caracterizat printr-un nume. El poate fi explicit sau implicit.

Tabela/relația este un subansamblu al produsului cartezian al mai multor domenii, caracterizat printr-un nume, prin care se definesc atributele ce aparțin aceleaşi clase de entități.

Atributul este coloana unei tabele, caracterizată printr-un nume.

Cheia este un atribut sau un ansamblu de atribute care au rolul de a identifica un tuplu dintr-o tabelă. *Tipuri* de chei: primare/alternate, simple/comune, externe.

Tuplul este linia dintr-o tabelă şi nu are nume. Ordinea liniilor (tupluri) şi coloanelor (atribute) dintr-o tabelă nu trebuie să prezinte nici-o importanță.

Schema tabelei este formată din numele tabelei, urmat între paranteze rotunde de lista atributelor, iar pentru fiecare atribut se precizează domeniul asociat.

Schema bazei de date poate fi reprezentată printr-o diagramă de structură în care sunt puse în evidență şi legăturile dintre tabele. Definirea legăturilor dintre tabele se face *logic* construind asocieri între tabele cu ajutorul unor atribute de legătură. Atributele implicate în realizarea legăturilor se găsesc fie în tabelele asociate, fie în tabele distincte construite special pentru legături. Atributul din tabela inițială se numeşte *cheie externă* iar cel din tabela finală este *cheie primară*. Legăturile posibile sunt *1:1, 1:m, m:n*. Potențial, orice tabelă se poate lega cu orice tabelă, după orice atribute.

Legăturile se stabilesc la momentul descrierii datelor prin limbaje de descriere a datelor (LDD), cu ajutorul restricțiilor de integritate. Practic, se stabilesc şi legături dinamice la momentul execuției.

2) **Operatorii modelului relațional** sunt operatorii din algebra relațională şi operatorii din calculul relațional.

Algebra relațională este o colecție de operații formale aplicate asupra tabelelor (relațiilor), şi a fost concepută de E.F.Codd. Operațiile sunt aplicate în *expresiile algebrice relaționale* care sunt cereri de regăsire. Acestea sunt compuse din operatorii relaționali şi operanzi. *Operanzii* sunt întotdeauna tabele (una sau mai multe). *Rezultatul* evaluării unei expresii relaționale este format dintr-o singură tabelă.

Algebra relațională are cel puțin puterea de regăsire a calcului relațional. O expresie din calculul relațional se poate transforma într-una *echivalentă* din algebra relațională şi invers.

Codd a introdus şase operatori *de bază* (reuniunea, diferența, produsul cartezian, selecția, proiecția, joncțiunea) şi doi operatori *derivați* (intersecția şi diviziunea). Ulterior au fost introduşi şi alți operatori derivați (*speciali*). În acest context, o*peratorii* din algebra relațională pot fi grupați în două categorii: pe mulțimi şi speciali.

Operatori pe mulțimi (R1, R2, R3 sunt relații (tabele)) *sunt:*

- *Reuniunea*. R3 = R1 ∪ R2, unde R3 va conține tupluri din R1 sau R2 luate o singură dată;
- *Diferența.* R3 = R1 \ R2, unde R3 va conține tupluri din R1 care nu se regăsesc în R2;
- *Produsul cartezian.* R3 = R1 × R2, unde R3 va conține tupluri construite din perechi (x1x2), cu x1∈R1 şi x2∈R2;
- *Intersecția.* R3 = R1 ∩ R2, unde R3 va conține tupluri care se găsesc în R1 şi R2 în acelaşi timp, etc.

Operatori relaționali speciali sunt:

- *Selecția*. Din R1 se obține o subtabelă R2, care va conține o submulțime din tuplurile inițiale din R1 ce satisfac un predicat (o condiție). Numărul de atribute din R2 este egal cu numărul de atribute din R1. Numărul de tupluri din R2 este mai mic decât numărul de tupluri din R1.
- *Proiecția.* Din R1 se obține o subtabelă R2, care va conține o submulțime din atributele inițiale din R1 şi fără tupluri duplicate. Numărul de atribute din R2 este mai mic decât numărul de atribute din R1.
- *Joncțiunea* este o derivație a produsului cartezian, ce presupune utilizarea unui calificator care să permită compararea valorilor

unor atribute din R1 şi R2, iar rezultatul în R3. R1 şi R2 trebuie să aibă unul sau mai multe atribute comune care au valori comune.

Algebra relaţională este prin definiţie *neprocedurală* (descriptivă), iar calculul relaţional permite o manieră de căutare *mixtă* (procedurală/neprocedurală).

Calculul relaţional se bazează pe *calculul predicatelor* de ordinul întâi (domeniu al logicii) şi a fost propus de E.F. Codd. *Predicatul* este o relaţie care se stabileşte între anumite elemente şi care poate fi confirmată sau nu. *Predicatul de ordinul 1* este o relaţie care are drept argumente variabile care nu sunt predicate. *Variabila* poate fi de tip tuplu (valorile sunt dintr-un tuplu al unei tabele) sau domeniu (valorile sunt dintr-un domeniu al unei tabele). *Cuantificatorii* (operatorii) utilizaţi în calculul relaţional sunt: universal (∀) şi existenţial (∃).

Construcţia de bază în calculul relaţional este expresia relaţională de calcul tuplu sau domeniu (funcţie de tipul variabilei utilizate).

Expresia relaţională de calcul este formată din: operaţia de efectuat, variabile (tuplu respectiv domeniu), condiţii (de comparaţie, de existenţă), formule bine definite (operanzi-constante, variabile, funcţii, predicate; operatori), cuantificatori.

Pentru implementarea acestor operatori există *comenzi specifice* în limbajele de manipulare a datelor (LMD) din sistemele de gestiune a bazelor de date relaţionale (SGBDR). Aceste comenzi sunt utilizate în operaţii de regăsire (interogare).

După **tehnica folosită** la manipulare, LMD sunt bazate pe:

- calculul relaţional (QUEL în Ingres, ALPHA propus de Codd);
- algebra relaţională (ISBL, RDMS);
- transformare (SQL, SQUARE);
- grafică (QBE, QBF).

Transformarea oferă o putere de regăsire *echivalentă* cu cea din calculul şi algebra relaţională. Se bazează pe *transformarea* (mapping) unui atribut sau grup de atribute într-un atribut dorit prin intermediul unor relaţii. *Rezultatul* este o relaţie (tabelă) care se poate utiliza într-o altă transformare.

Grafica oferă *interactivitate* mare pentru constrirea cererilor de regăsire. Utilizatorul specifică cerea *alegând* sau completând un ecran structurat grafic. Poate fi *folosit* de către toate categoriile de utilizatori în informatică.

3) **Restricţiile de integritate ale modelului relaţional** sunt structurale şi comportamentale.

Restricțiile structurale sunt:

- *Restricția de unicitate a cheii.* Într-o tabelă nu trebuie să existe mai multe tupluri cu aceeaşi valoare pentru ansamblul cheie;
- *Restricția referențială.* Intr-o tabelă t1 care referă o tabelă t2, valorile cheii externe trebuie să figureze printre valorile cheii primare din t2 sau să ia valoarea null (neprecizat);
- *Restricția entității.* Intr-o tabelă, atributele din cheia primară nu trebuie să ia valoarea NULL.

Cele trei restricții de mai sus sunt *minimale*.

Pe lângă acestea, există o serie de alte restricții structurale care se referă la dependențele dintre date: funcționale, multivaloare, joncțiune etc. (sunt luate în considerare la tehnicile de proiectare a bazelor de date relaționale - BDR).

Restricțiile de comportament sunt cele care se definesc prin comportamentul datelor şi țin cont de valorile din BDR:

- Restricția *de domeniu.* Domeniul corespunzător unui atribut dintr-o tabelă trebuie să se încadreze între anumite valori;
- Restricții *temporare*. Valorile anumitor atribute se compară cu nişte valori temporare (rezultate din calcule etc.).

Restricțiile de comportament fiind foarte generale se gestionează fie la momentul descrierii datelor (de exemplu prin clauza CHECK), fie în afara modelului la momentul execuției.

Restricțiile de integritate suportate de Oracle sunt:

- NOT NULL nu permite valori NULL în coloanele unei tabele;
- UNIQUE nu sunt permise valori duplicat în coloanele unei tabele;
- PRIMARY KEY nu permite valori duplicate sau NULL în coloana sau coloanele definite astfel;
- FOREIGN KEY presupune ca fiecare valoare din coloana sau setul de coloane defini astfel să aibă o valoare corespondentă identică în tabela de legătură, tabelă în care coloana corespondentă este definită cu restricția UNIQUE sau PRIMARY KEY;
- CHECK elimină valorile care nu satisfac anumite cerințe (condiții) logice.

Termenul de chei (keys) este folosit pentru definirea câtorva categorii de constrângeri şi sunt: *primary key, unique key, foreign key, referenced key.*

Se consideră că modelul relațional are o serie de limite cum ar fi:

- Simplitatea modelului îl face dificil de aplicat pentru noile tipuri

de aplicații (multimedia, internet etc.);
- Nu asigură o independență logică totală a datelor de aplicație;
- Poate crește redundanța datelor.

1.2. BAZE DE DATE RELAȚIONALE

Bazele de date relaționale (BDR) utilizează modelul de date relațional şi noțiunile aferente. BDR au o solidă fundamentare teoretică, în special prin cercetările de la IBM conduse de E.F.Codd.

BDR este un ansamblu organizat de *tabele* (relații) împreună cu *legăturile* dintre ele.

Concepte utilizate la organizarea datelor în BDR şi respectiv fişiere sunt prezentate în tabelul 1.1.

Concepte utilizate în organizarea datelor

Tabelul 1.1.

Fişiere	fişier	înregistrare	câmp	valori
BDR	tabelă(relație)	tuplu (linie)	atribut(coloană)	domeniu valori

Avantajele BDR față de fişiere sunt prezentate în tabelul 1.2.

Avantajele BDR

Tabelul 1.2.

CRITERIU	BDR	FIŞIERE
Independența datelor	logică şi fizică	fizică
Niveluri de structurare	conceptual, logic şi fizic	logic şi fizic
Deschidere şi portabilitate	mare	mică
Reprezentarea şi utilizarea datelor	simplificat prin model	complicat
Structura de date se păstrează	în dicționarul BDR	în programe.

Atunci când dorim să realizăm o bază de date relațională trebuie să ştim clar ce avem de făcut, adică să stabilim obiectivele activității noastre. În acest sens, câteva dintre cele mai importante **obiective**, le prezentăm în continuare:

- *Partiționarea* semnifică faptul că aceleaşi date trebuie să poată fi folosite în moduri diferite de către diferiți utilizatori;
- *Deschiderea* se referă la faptul că datele trebuie să fie uşor adaptabile la schimbările care pot apărea (actualizarea structurii, tipuri noi de date etc.);
- *Eficiența* are în vedere stocarea şi prelucrarea datelor, care trebuie

să se facă la costuri cât mai scăzute, costuri care să fie inferioare beneficiilor obținute;

- *Reutilizarea* înseamnă faptul că fondul de date existent trebuie să poată fi reutilizat în diferite aplicații informatice;
- *Regăsirea* este o actvitate frecventă pe bazele de date şi de aceea cererile de regăsire trebuie să poată fi adresate uşor de către toate categoriile de utilizatori, după diferite criterii;
- *Accesul* înseamnă modul de localizare a datelor şi acest lucru trebuie să poată fi realizat prin diferite moduri de acces, rapid şi uşor;
- *Modularizarea* presupune faptul că realizarea BDR trebuie să se poată face modular pentru generalitate şi posibilitatea lucrului în echipă;
- *Protecția* bazei de date trebuie asigurată sub ambele aspecte: securitatea şi integritatea datelor;
- *Redundanța* se asigură în limite acceptabile prin implementarea unui model de date pentru baze de date şi prin utilizarea unei tehnici de proiectare a BDR. Se asigură astfel, o redundanță minimă şi controlată;
- *Independența* datelor față de programe trebuie asigurată atât la nivel logic cât şi şi fizic.

Bazele de date relaționale au evoluat ca un tip special de aplicații informatice, şi anume cele care au organizarea datelor în memoria externă conform unui model de date specific. De aceea, în metodologia de realizare a BDR se parcurg, în cea mai mare parte, cam aceleaşi etape ca la realizarea unei aplicații informatice, cu o serie de aspecte specifice. Pe de altă parte, în literatura de specialitate, sunt diferite propuneri de metodologii de realizare a bazelor de date.

Ținând cont de cele două aspecte de mai sus, sunt propuse câteva actvivități care trebuie parcurse la realizarea unei baze de date. Aceste activități vor fi regăsite, sub aceeaşi denumire sau sub denumiri diferite, în majoritatea metodologiilor de realizare a bazelor de date, din literatura de specialitate.

Activitățile (etapele) parcurse pentru realizarea unei BDR sunt: analiza de sistem, proiectarea noului sistem, realizarea componentelor logice, punerea în funcțiune, dezvoltarea.

1) Scopul analizei de sistem este de a evidenția cerințele aplicației şi resursele utilizate (studiul), precum şi de a evalua aceste cerințe prin modelare (analiza).

Studiul situației existente se realizează prin: definirea caracteristicilor generale ale unității, identificarea activităților desfăşurate, identificarea resurselor existente (informaționale, umane, energetice, echipamente, financiare etc.), identificarea necesităților de prelucrare.

Analiza este o activitate de modelare (conceptuală) şi se realizează sub trei aspecte: structural, dinamic şi funcțional.

a) Analiza structurală evidențiază, la nivel conceptual, modul de structurare a datelor şi a legăturilor dintre ele. Cea mai utilizată tehnică este *entitate-asociere*. Aceasta conține:

- Identificarea entităților: fenomene, procese, obiecte concrete sau abstracte (substantivele din prezentarea activității descrise) (exemple de entități: Persoane, Produse, Beneficiari).
- Identificarea asocierilor dintre entități ca fiind legăturile semnificative de un anumit tip (verbele din prezentarea activității descrise).
- Identificarea atributelor ce caracterizează fiecare entitate în parte (exemple de atribute: Marca, Nume, Adresă).
- Stabilirea atributelor de identificare unică a realizărilor entității, drept chei.

Rezultatul analizei structurale este modelul static (structural) numit şi diagrama entitate-asociere. Diagrama entitate-asociere (Entity-Relationship) poate fi generată cu produse software tip CASE (Computer Aided Software Engineering), ca de exemplu Oracle Designer. Pornind de la o astfel de diagramă, se pot construi, în actvitatea de proiectare, schemele relațiilor (tabelelor).

b) Analiza dinamică evidențiază comportamentul elementelor sistemului la anumite evenimente. Una din tehnicile utilizate este diagrama *stare-tranziție*. Aceasta presupune:

- Identificarea stărilor în care se pot afla componentele sistemului.
- Identificarea evenimentelor care determină trecerea unei componente dintr-o stare în alta.
- Stabilirea tranzițiilor admise între stări.
- Construirea diagramei stare-tranziție.

Rezultatul analizei dinamice este modelul dinamic.

c) Analiza funcțională evidențiază modul de asigurare a cerințelor informaționale (fluxul prelucrărilor) din cadrul sistemului, prin care intrările sunt transformate în ieşiri. Cea mai utilizată tehnică este *diagrama de flux* al datelor. Conform acestei tehnici se delimitează:

- Aria de cuprindere a sistemului.
- Se identifică sursele de date.

- Se identifică modul de circulaţie şi prelucrare a datelor.
- Se identifică apoi rezultatele obţinute.

Rezultatul analizei funcţionale este modelul funcţional.

2) Proiectarea structurii bazei de date se face pe baza modelelor realizate în activitatea de analiză. Inainte de proiectarea bazei de date se alege tipul de sistem de gestiune a bazei de date. Alegerea SBGD-ului se face ţinând cont de două aspecte: cerinţele aplicaţiei (utilizatorului) şi performanţele tehnice ale SGBD-ului.

Cerinţele aplicaţiei se referă la: volumul de date estimat a fi memorat şi prelucrat în BDR; complexitatea problemei de rezolvat; ponderea şi frecvenţa operaţiilor de intrare/ieşire; condiţiile privind protecţia datelor; operaţiile necesare (încărcare/validare, actualizare, regăsire etc.); particularităţile activităţii pentru care se realizează baza de date.

Performanţele tehnice ale SGBD-ului se referă la: modelul de date pe care-l implementează; ponderea utilizării SGBD-ului pe piaţă şi tendinţa; configuraţia de calcul minimă cerută; limbajele de programare din SGBD; facilităţile de utilizare oferite pentru diferite categorii de utilizatori; limitele SGBD-ului; optimizările realizate de SGBD; facilităţile tehnice; lucrul cu mediul distribuit şi concurenţa de date; elementele multimedia; instrumentele CASE; interfeţele de comunicare; posibilitatea de autodocumentare; instrumentele specifice oferite.

Proiectarea BDR se realizează prin proiectarea schemelor BDR şi proiectarea modulelor funcţionale specializate.

Schemele bazei de date sunt: conceptuală, externă şi internă.

a) Proiectarea schemei conceptuale porneşte de la identificarea setului de date necesar sistemului. Aceste date sunt apoi integrate şi structurate într-o schemă (exemplu: pentru BDR relaţionale cea mai utilizată tehnică este *normalizarea*). Pentru acest lucru se parcurg paşii:

- Stabilirea schemei conceptuale iniţiale care se deduce din modelul entitate-asociere (vezi analiza structurală). Pentru acest lucru, se transformă fiecare entitate din model într-o colecţie de date (fişier), iar pentru fiecare asociere se definesc cheile aferente. Dacă rezultă colecţii izolate, acestea se vor lega de alte colecţii prin chei rezultând asocieri (*1:1, 1:m, m:n).*
- Ameliorarea progresivă a schemei conceptuale prin eliminarea unor anomalii (exemplu: cele cinci forme normale pentru BDR relaţionale).
- Stabilirea schemei conceptuale finale trebuie să asigure un echilibru între cerinţele de actualizare şi performanţele de exploatare (exemplu: o formă normală superioară asigură

performanțe de actualizare, dar timpul de răspuns va fi mai mare).

Tehnica de normalizare este utilizată în activitatea de proiectare a structurii BDR şi constă în eliminarea unor anomalii (neajunsuri) de actualizare din structură.

Anomaliile de actualizare sunt situații nedorite care pot fi generate de anumite tabele în procesul proiectării lor:

- *Anomalia de ştergere* semnifică faptul că stergând un tuplu dintr-o tabelă, pe lângă informațiile care trebuie şterse, se pierd şi informațiile utile existente în tuplul respectiv;
- *Anomaliile de adăugare* semnifică faptul că nu pot fi incluse noi informații necesare într-o tabelă, deoarece nu se cunosc şi alte informații utile (de exemplu valorile pentru cheie);
- *Anomalia de modificare* semnifică faptul că este dificil de modificat o valoare a unui atribut atunci când ea apare în mai multe tupluri.

Normalizarea este o teorie construită în jurul conceptului de *forme normale* (FN), care ameliorează structura BDR prin înlăturarea treptată a unor neajunsuri şi prin imprimarea unor facilități sporite privind manipularea datelor.

Normalizarea utilizează ca metodă *descompunerea* (top-down) unei tabele în două sau mai multe tabele, păstrând informații (atribute) de legătură.

FN1. O tabelă este în FN1 dacă toate atributele ei conțin valori elementare (nedecompozabile), adică fiecare tuplu nu trebuie să aibă date la *nivel de grup* sau *repetitiv*. Structurile de tip arborescent şi rețea *se transformă* în tabele cu atribute elemntare.

O tabelă în FN1 prezintă încă o serie de *anomalii* de actualizare datorită eventualelor dependențe funcționale incomplete.

Fiecare structură repetitivă generează (prin descompunere) o nouă tabelă, iar atributele la nivel de grup se înlătură, rămânând doar cele elemntare.

FN2. O tabelă este în FN2 dacă şi numai dacă este în FN1 şi fiecare atribut noncheie al tabelei este *dependent funcțional complet* de cheie. Un atribut B al unei tabele *depinde funcțional* de atributul A al aceleiaşi tabele, dacă fiecărei valori a lui A îi corespunde o singură valoare a lui B, care îi este asociată în tabelă. Un atribut B este *dependent funcțional complet* de un ansamblu de atribute A în cadrul aceleiaşi tabele, dacă B este dependent funcțional de întreg ansamblul A (nu numai de un atribut din ansamblu).

O tabelă în FN2 prezintă încă o serie de *anomalii* de actualizare, datorită eventualelor dependențe tranzitive.

Eliminarea dependențelor incomplete se face prin descompunerea tabelei inițiale în două tabele, ambele conținând atributul intermediar (B).

FN3. O tabelă este în FN3 dacă şi numai dacă este în FN2 şi fiecare atribut noncheie depinde în mod *netranzitiv* de cheia tabelei. Într-o tabelă T, fie A,B,C trei atribute cu A cheie. Dacă B depinde de A (A → B) şi C depinde de B (B → C) atunci C depinde de A în mod *tranzitiv*. *Eliminarea* dependențelor tranzitive se face prin descompunerea tabelei inițiale în două tabele, ambele conținând atributul intermediar (B).

O tabelă în FN3 prezintă încă o serie de *anomalii* de actualizare, datorate eventualelor dependențe multivaloare.

O definiție mai riguroasă pentru FN3 a fost dată prin forma intermediară **BCNF** (Boyce Codd Normal Form): o tabelă este în BCNF dacă fiecare determinant este un candidat cheie.Determinantul este un atribut elementar sau compus față de care alte atribute sunt complet dependente funcțional.

FN4. O tabelă este în FN4 dacă şi numai dacă este în FN3 şi nu conține două sau mai multe dependențe *multivaloare*. Într-o tabelă T, fie A,B,C trei atribute. În tabela T se menține dependența multivaloare A dacă şi numai dacă mulțimea valorilor lui B ce corespunde unei perechi de date (A,C), depinde numai de o valoare a lui A şi este independentă de valorile lui C.

FN5. O tabelă este în FN5 dacă şi numai dacă este în FN4 şi fiecare *dependență joncțiune* este generată printr-un candidat cheie al tabelei. În tabela T (A,B,C) se menține *dependența joncțiune* (AB, AC) dacă şi numai dacă T menține dependența multivaloare A -->> B sau C.

Dependența multivaloare este caz particular al dependenței joncțiune. Dependența funcțională este caz particular al dependenței multivaloare.

b) Proiectare schemei externe are rolul de a specifica viziunea fiecărui utilizator asupra BDR. Pentru acest lucru, din schema conceptuală se identifică datele necesare fiecărei viziuni. Datele obținute se structurează logic în subscheme ținând cont de facilitățile de utilizare şi de cerințele utilizator. Schema externă devine operațională prin construirea unor viziuni (view) cu SGBD-ul şi acordarea drepturilor de acces. Datele într-o viziune pot proveni din una sau mai multe colecții şi nu ocupă spațiul fizic.

c) Proiectarea schemei interne presupune stabilirea structurilor de memorare fizică a datelor şi definirea căilor de acces la date. Acestea sunt specifice fie SGBD-ului (scheme de alocare), fie sistemului de operare. Proiectarea schemei interne înseamnă estimarea *spațiului fizic* pentru BDR, definirea unui model fizic de alocare (a se vedea dacă SGBD-ul permite explicit acest lucru) şi definirea unor *indecşi* pentru accesul direct, după

cheie, la date.

Proiectarea modulelor funcționale ține cont de concepția generală a BDR, precum și de schemele proiectate anterior. În acest sens, se proiectează fluxul informațional, modulele de încărcare și manipulare a datelor, interfețele specializate, integrarea elementelor proiectate cu organizarea și funcționarea BDR.

3) Realizarea componentelor logice. Componentele logice ale unei BD sunt programele de aplicație dezvoltate, în cea mai mare parte, în SGBD-ul ales. Programele se realizează conform modulelor funcționale proiectate în etapa anterioară. Componentele logice țin cont de ieşiri, intrări, prelucrări și colecțiile de date. În paralel cu dezvoltarea programelor de aplicații se întocmesc și documentațiile diferite (tehnică, de exploatare, de prezentare).

4) Punerea în funcțiune și exploatarea. Se testează funcțiile BDR mai întâi cu date de test, apoi cu date reale. Se încarcă datele în BDR și se efectuează procedurile de manipulare, de către beneficiar cu asistența proiectantului. Se definitivează documentațiile aplicației. Se intră în exploatare curentă de către beneficiar conform documentației.

5) Dezvoltarea sistemului. Imediat după darea în exploatare a BDR, în mod continuu, pot exista factori perturbatori care generează schimbări în BDR. Factorii pot fi: organizatorici, datorați progresului tehnic, rezultați din cerințele noi ale beneficiarului, din schimbarea metodologiilor etc.

1.3. DEFINIREA SISTEMULUI DE GESTIUNE A BAZELOR DE DATE RELAȚIONALE (SGBDR)

Teoria relațională, foarte bine pusă la punct într-un domeniu de cercetare distinct, a dat o fundamentare solidă realizării de SGBD-uri performante. La sfârșitul anilor 80 și apoi în anii 90 au apărut, în special o dată cu pătrunderea în masă a microcalculatoarelor, numeroase SGBDR-uri. Aceasta a însemnat o evoluție de la SGBD-urile de generația întâi (arborescente și rețea) spre cele de generația a doua (relaționale). Această evoluție s-a materializat, în principal în: oferirea de limbaje de interogare neprocedurale, îmbunătățirea integrității și securității datelor, optimizarea și simplificarea acceselor.

Teoria relațională este un ansamblu de concepte, metode și instrumente care a dat o fundamentare riguroasă realizării de SGBDR performante.

Paralela între conceptele utilizate în evoluția organizării datelor în memoria externă până la sistemele relaționale este prezentata in tabelul 1.3:

Tabelul 1.3

FIŞIERE	TEORIA BD	TEORIA RELAȚIONALĂ	SGBDR
Fişier	Colecție de date	Relație	Tabela
Înregistrare	Familie de caracteristici	Tuplu	Linie
Câmp	Caracteristică	Atribut	Coloană
Valoare	Domeniu de valori	Domeniu	Domeniu

Regulile lui Codd

E.F. Codd (cercetător la IBM) a formulat 13 reguli care exprimă cerințele maximale pentru ca un SGBD să fie relațional. Regulile sunt utile pentru evoluarea performanțelor unui SGBDR. Acestea sunt:

R_0. Gestionarea datelor la nivel de relație: limbajele utilizate trebuie să opereze cu relații (unitatea de informație).

R_1. Reprezentarea logică a datelor: toate informațiile din BDR trebuie stocate şi prelucrate ca tabele.

R_2. Garantarea accesului la date: LMD trebuie să permită accesul la fiecare valoare atomică din BDR (tabelă, coloană, cheie).

R_3. Valoarea NULL: trebuie să se permită declararea şi prelucrarea valorii NULL ca date lipsă sau inaplicabile.

R_4. Metadatele: informațiile despre descrierea BDR se stochează în dicționar şi tratează ca tabele ,la fel ca datele propiu-zise.

R_5. Limbajele utilizate: SGBDR trebuie să permită utilizarea mai multor limbaje, dintre care cel puțin unul să permită definirea tabelelor (de bază şi virtuale), definirea restricțiilor de integritate, manipularea datelor, autorizarea accesului, tratarea tranzacțiilor.

R_6. Actualizarea tabelelor virtuale: trebuie să se permită ca tabelele virtuale să fie şi efectiv actualizabile, nu numai teoretic actualizabile (exemplu atributul “valoare” dintr-o tabelă virtuală nu poate fi actualizat).

R_7. Actualizările în baza de date: manipularea unei tabele trebuie să se facă prin operații de regăsire dar şi de actulizare.

R_8. Independența fizică a datelor: schimbarea stucturii fizice a datelor (modul de reprezentare (organizare) şi modul de acces) nu afectează programele.

R_9. Independența logică a datelor: schimbarea structurii de date (logice) a tabelelor nu afectează programele.

R_{10}. Restricțiile de integritate: acestea, trebuie să fie definite prin LDD şi stocate în dicționarul (catalogul) BDR.

R_{11}. Distribuirea geografică a datelor: LMD trebuie să permită ca

programele de aplicație să fie aceleaşi atât pentru date distribuite cât şi petru date centralizate (alocarea şi localizarea datelor vor fi în sarcina SGBDR-ului).

R_{12}. Prelucrarea datelor la nivel de bază (scăzut): dacă SGBDR posedă un limbaj de nivel scăzut (prelucrarea datelor se face la nivel de înregistrare), acesta nu trebuie utilizat pentru a evita restricțiile de integritate.

Regulile lui Codd pot fi grupate, conform cerințelor exprimate în cinci categorii, conform tabelului 1.4.

Tabelul 1.4. Gruparea regulilor lui Codd

	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}
1.Reguli de bază (funda-mentale)	da												da
2.Reguli structurale		da					da						
3.Reguli privind integritatea datelor				da							da		
4.Reguli privind manipularea datelor			da		da	da		da					
5.Reguli privind independența datelor									da	da		da	

Regulile lui Codd sunt greu de indeplinit în totalitate de către SGBDR. Pornind de la cele 13 reguli de mai sus, au fost formulate o serie de criterii (cerințe) pe care trebuie să le îndeplinească un SGBD pentru a putea fi considerat relațional într-un anumit grad. S-a ajuns astfel, la mai multe grade de relațional pentru SGBDR: *cu interfață relațională* (toate datele se reprezintă în tabele, există operatorii de selecție, proiecție şi joncțiune doar pentru interogare), *pseudorelațional* (toate datele se reprezintă în tabele, există operatorii de selecție, proiecție şi joncțiune fără limitări), *minimal relațional* (este pseudorelațional şi în plus, operațiile cu tabele nu fac apel la pointeri observabili de utilizatori), *complet relațional* (este minimal relațional şi în plus, există operatorii de reuniune, intersecție şi diferență, precum şi restricțiile de integritate privind unicitatea cheii şi restricția referențială).

În concluzie, ***SGBDR*** **este un sistem software complet care implementeză modelul de date relațional şi respectă cerințele impuse de acest model.** El este o interfață între utilizatori şi baza de date.

1.4. CARACTERIZAREA SGBDR

Sistemele relaționale îndeplinesc funcțiile unui SGBD cu o serie de aspecte specifice care rezultă din definirea unui SGBDR.

Caracterizarea SGBDR se poate face pe două niveluri: global (sistemele relaționale sunt privite ca o categorie distinctă de SGBD) şi particular (fiecare SGBDR are aspecte individuale comparativ cu altele similare).

A. Mecanismele şi instrumentele care ajută la **caracterizarea globală** a SGBDR-urilor sunt: limbajele relaționale, protecția datelor, optimizarea cererilor de regăsire, utilitarele specializate.

1) Limbajele relațional

SGBDR oferă seturi de comenzi pentru descrierea şi manipularea datelor. Acestea pot fi incluse într-un singur limbaj relațional (SQL, QUEL, QBE, SQUARE, ALPHA, ISBL) sau separate în LDD şi LMD. În ambele situații, comenzile pentru definirea datelor sunt distincte de cele pentru manipularea datelor.

Limbajele relaționale de definire a datelor (LDD) sunt simplificate, cu puține comenzi. Descrierea datelor este memorată în BDR, sub formă de tabele, în dicționarul (metabaza) bazei de date. Facilități de descriere sunt prezente în SGBDR prin comenzi, care definesc anumite operații, la nivelurile: conceptual, logic, fizic.

Limbajele relaționale de manipulare a datelor (LMD) pot fi caracterizate după criterii generale, funcționale şi calitative.

a) Caracterizarea generală a LMD se face după modul de tratare a datelor, operatorii relaționali, realizatorii şi utilizatorii limbajului.

Modul de tratare a datelor. Toate LMD relaționale realizează o tratare la nivel de *ansamblu* a datelor: unitatea de informative pentru lucru este *tabela*. Avantajele sunt date de posibilitatea gestionării automat a tuplurilor duplicate şi prelucrarea paralelă a ansamblurilor.

La comunicarea unui LMD relational cu un limbaj universal, avantajele se pierd deoarece comunicarea se poate face doar tuplu cu tuplu şi nu la nivel de ansamblu. Deoarece limbajele universale oferă alte avantaje legate de proceduralitate, soluția este de a integra în acestea un limbaj relațional. *Cursorul* este soluția în SGBDR pentru a face trecerea de la tratarea la nivel de ansamblu la cea la nivel de înregistrare (tuplu).

Operatorii relaționali implementați. SGBDR s-au dezvoltat, din punct de vedere relațional, având la bază calculul relațional orientat pe tuplu (ALPHA, QUEL), calculul relațional orientat pe domeniu (QBE), algebra relațională (ISBL), transformarea (mapping) (SQL, SQUARE). Limbajele bazate pe calculul relațional sunt neprocedurale, cele bazate pe algebra

relațională sunt procedurale, celelalte sunt combinații.

Realizatorii limbajelor relaționale s-au orientat pe domenii precise din teoria relațională. Astfel, au rezultat: limbaje relaționale standardizate internațional (exemplu SQL - ANSI), limbaje cu standard de utilizare impus de constructor (exemplu QUEL), limbaje nestandardizate (celelalte limbaje relaționale).

Utilizatorii limbajelor relaționale sunt mult diversificați. SGBDR oferă atât elemente procedurale (pentru specialişti) cât şi neprocedurale (pentru nespecialilşti).

b) Caracterizarea funcțională a LMD se face după facilitățile de interogare, actualizare a datelor, etc.

Facilitățile de interogare a datelor. Acestea sunt puternice şi oferite prin comenzi pentru interogarea tabelelor de bază (exemplu SELECT) şi interogarea tabelelor virtuale (exemplu SELECT).

Facilitățile de actualizare a datelor. Acestea se referă la actualizarea tabelelor de bază şi a tabelelor virtuale prin comenzile: INSERT INTO (adaugă rânduri la sfârşitul unei tabele); UPDATE (modifică rânduri dintr-o tabelă); DELETE FROM (şterge rânduri dintr-o tabelă). Unele SGBDR nu permit actualizarea tabelelor virtuale (exemplu Foxpro), altele permit acces lucru cu o serie de restricții pentru ca operația să se propage spre tabelele de bază fără ambiguități (exemplu DB2, Oracle).

Alte facilități funcționale. La facilitățile relaționale de mai sus, SGBDR-urile oferă şi alte facilități pe care le au toate limbajele de programare procedurale cum sunt: calculul aritmetic prin operatorii specifici (+, -, *, /, **); agregarea prin funcții standard (SUM) şi prin comenzi (COMPUTE OF expr); comenzi de intrare/ieşire standard (ACCEPT…PROMPT…).

c) Caracterizarea calitativă a LMD se face după puterea selectivă, uşurința de învățare, utlizare şi eficiența limbajului.

Puterea selectivă a LMD relaționale este dată de posibilitatea selectării datelor după criterii (filtre) complexe (exemplu comanda SELECT).

Uşurința de învățare şi utilizare este nuanțată în funcție de tipul LMD-ului relațional. Cele bazate pe calculul relațional sunt neprocedurale (descriptive), deci uşor de învățat şi utilizat (apropiat, ca stil, de limbajul natural) (exemplu QUEL) iar cele bazate pe algebra relațională sunt procedurale (algoritmice), deci mai greu de învățat şi utilizat (exemplu ISBL). Cele intermediare promovează stilul neprocedural dar acceptă şi elemente de control procedural (exemplu SQL) iar cele bazate pe grafică oferă primitive grafice pentru machetarea cererilor de regăsire, deci uşor de utilizat (exemplu QBE).

Eficiența utilizării este determinată de posibilitatea optimizării cererilor de regăsire. LMD bazate pe calculul relațional lasă compilatorul să aleagă ordinea de execuție a operațiilor, deci rezultă o eficiența mare. LMD bazate pe algebra relațională au o ordine impusă pentru execuția operațiilor, deci rezultă o eficiență mica.

2) Protecția datelor

Aspectele privind protecția datelor sunt foarte importante pentru un sistem de bază de date şi ele trebuie implementate de către SGBDR. Protecția bazei de date se referă la integritatea datelor (integritatea semantică, concurența la date, salvarea/restaurarea) şi securitatea datelor (autorizarea accesului, viziunile, procedurile speciale, criptarea).

a) Integritatea semantică. Definirea restricțiilor de integritate se face, conform cerințelor modelului relațional, în LDD (exemplu CREATE TABLE, ALTER TABLE). Utilizarea restricțiilor de integritate se face cu ajutorul unor mecanisme care controlează validitatea regulilor pentru fiecare nouă stare a BD. Aceste mecanisme sunt metode de detectare a inconsistenței datelor (se verifică restrciţiile de integritate) la sfârşitul tranzacţiilor, care se realizează automat de SGBDR şi puncte de verificare a integrității fixate de utilizator, acolo unde doreşte el în program.

b) Concurența la date (coerența). Unitatea de lucru pentru asigurarea coerenței datelor este *tranzacția*. Aceasta este un ansamblu de comenzi tratate unitar. Tranzacția se execută în totalitate sau deloc. Coerența poate fi afectată la actualizarea concurentă sau la incidente.

Mecanismele utilizate de SGBDR pentru asigurarea coerenței datelor (controlul accesului concurent) sunt:

- Blocarea la diferite niveluri: bază de date, tabelă, tuplu, atribut;
- Definirea unor puncte de salvare în interiorul tranzacţiilor (exemplu comanda savepoint);
- Tranzacţii explicite (begin şi end transaction) şi implicite (comenzile de actualizare);
- Fişiere jurnal.

3) Optimizarea regăsirii

Cererile de regăsire se exprimă în SGBDR în diferite limbaje relaționale. Pentru a se obține un rezultat optim, se utilizează interfețe automate de rescriere a cererilor de regăsire, prin parcurgerea a doi paşi:

- Exprimarea cererilor de regăsire sub forma unor expresii algebrice relaționale, care are la bază echivalența dintre calculul şi algebra relațională .
- Aplicarea unor transformări algebrice relaționale asupra expresiilor construite în pasul anterior, pentru a se obține expresii

relaţionale echivalente şi eficiente.

Transformarea se poate realiza prin doua *strategii de optimizare*: generale, specifice.

Strategiile generale sunt independente de modul de memorare a datelor. Ele se bazează pe propietăţile operaţiilor din algebra relaţională (comutativitatea, asociativitatea, compunerea). Astfel de strategii sunt: selecţia înaintea joncţiunii, proiecţia înaintea joncţiunii, selecţia înaintea proiecţiei, combinarea selecţiei multiple.

Strategiile specifice ţin cont de modul de memorare a datelor şi ele sunt caracteristice unui SGBDR. Elementele care influenţează executarea operaţiilor ce intervin la o cerere de regăsire sunt: accesul direct, reguli de ordonare a expresiilor algebrice specifice unui SGBDR.

4) Utilitarele specializate

Posibilitatile de utilizare ale unui SGBDR sunt influenţate de utilitarele specializate pe care le are, pentru diferitele categorii de utilizatori (în Oracle: Developer pentru dezvoltatori, Designer pentru analişti, Administration Tools şi Utilities pentru administrator etc.).

B**.** Pentru a face o **caracterizare particulară**,un anumit SGBDR vom lua în considerare o serie de criterii de comparaţie. Aceste criterii se vor urmări, grupate pe anumite categorii, pentru câteva SGBDR-uri care ne interesează. După această analiză vom avea un argument serios pentru a putea alege un SGBDR în scopul dezvoltării unei aplicaţii cu baze de date.

Gruparea caracteristicilor particulare de comparaţie a SGBDR-urilor o vom face în funcţie de facilităţile de descriere, manipulare, utilizare şi administrare a datelor.

Caracteristicile în funcţie de facilităţile de descriere sunt: modul de implementare a modelului relaţional; conceptul de bază de date utilizat în schemă; definirea metadatelor; definirea relaţiilor virtuale; actualizarea schemei relaţiei; restricţiile de integritate ce pot fi declarate.

Caracteristicile în funcţie de facilităţile de manipulare sunt: LMD relaţional implementat; funcţiile de calcul aritmetic şi funcţiile agregate; modurile de acces la date; programarea orientată-obiect; tratarea valorii NULL; optimizarea cererilor de regăsire; actualizarea relaţiilor de bază şi virtuale.

Caracteristicile în funcţie de facilităţile de utilizare şi administrare sunt: instrumentele de dezvoltare; instrumentele CASE; instrumentele analize statistice; software-ul pentru acces de la distanţă; utilitarele de întreţinere; mecanismele pentru autorizarea accesului la date.

1.5. EXEMPLE DE SISTEME DE GESTIUNE A BAZELOR DE DATE RELAȚIONALE

Oracle. Este realizat de firma Oracle Corporation USA. Sistemul este complet relațional, robust, se bazează pe SQL standard extins. Arhitectura sistemului este client/server, permțând lucrul, cu obiecte şi distribuit. Are BD Internet şi modul de optimizare a regăsirii. Ultima versiune este Oracle 10g.

DB2. Este realizat de firma IBM. Sistemul respectă teoria relațională, este robust şi se bazează pe SQL standard. Permite lucrul distribuit şi are modul de optimizare a regăsirii.

Informix. Este realizat de firma Informix, respectă teoria relațională şi permite lucru distribuit.

Progress. Este realizat de firma Progress Software. Are limbaj propriu (Progress 4GL) dar suportă şi SQL. Rulează pe o gamă largă de calculatoare sub diferite sisteme de operare.

SQL Server. Este realizat de firma Microsoft. Se bazează pe SQL şi rulează în arhitectura client/server.

Ingress II. Este realizat de firma Computer Associates. Este un SGBDR complet, implementează două limbaje relaționale (întâi QUEL şi apoi SQL) şi este suportat de diferite sisteme de operare (Windows, UNIX). Lucrează distribuit în arhitectura client/server, are extensie cu facilități orientate obiect şi permite aplicații de tip Internet. Organizarea fizică a tabelelor se face prin sistemul de operare.

Visual FoxPro. Este realizat de firma Microsoft. Are un limbaj procedural propiu foarte puternic, o extensie orientată obiect, programare vizuală şi nucleu extins de SQL.

Access. Este realizat de firma Microsoft. Se bazează pe SQL, are limbajul procedural gazdă (Basic Access) şi instrumente de dezvoltare.

Paradox. Este realizat de firma Borland. Are limbaj procedural propiu (PAL) şi suportă SQL.

CAPITOLUL 2.
FACILITATILE SI ARHITECTURA SISTEMULUI ORACLE

2.1. EVOLUȚIA ŞI FACILITĂȚILE SISTEMULUI ORACLE

Oracle este un sistem de gestiune a bazelor de date complet relațional, extins, cu facilități din tehnologia orientată obiect (OO). Sistemul Oracle este realizat de firma *Oracle Corporation* care a fost înființată în anul 1977 în SUA - California şi acum este cel mai mare furnizor de software de gestiunea datelor. Acesta este operațional pe toată gama de calculatoare (micro, mini, mainframe) sub diverse sisteme de operare.

Prima versiune de SGBD Oracle a fost realizată la sfârşitul anilor '70 respectând teoria relațională. În cadrul sistemului a fost implementat de la început *limbajul relațional SQL* pe care l-a dezvoltat ulterior față de versiunea standard rezultând *SQL*Plus.*

Începând cu *versiunea 5.0* SGBD Oracle are următoarele facilități suplimentare: funcționează în arhitectura client/server; are limbaj procedural propriu PL/SQL; are precompilatoare ca interfață cu limbajele universale.

În iunie 1997 s-a lansat *SGBD Oracle versiunea 8.0,* inclusiv în România, care a marcat o nouă generație de baze de date Oracle deoarece inițiază trecerea de la arhitectura client/server la arhitectura NC (Network Computing), are o mare deschidere, are optimizări performante şi pune accent mai mare pe analiză (modelare-funcționalitate) față de programare (codificare).

În noiembrie 1998 s-a lansat *SGBD Oracle 8i* ca sistem de baze de date pe Internet. Această versiune are următoarele caracteristici:

- Este reproiectat arhitectural în mod fundamental şi se încadrează în tendința de trecere de la arhitectura client/server la arhitectura NC;
- Permite dezvoltarea unei baze de date de orice dimensiune, în mod centralizat sau distribuit;
- Are facilități de salvare/restaurare automate şi inteligente;
- Permite partiționarea integrală pentru tabele şi indecşi;
- Are mesagerie integrală, prin comunicarea între aplicații şi procesare offline (chiar dacă aplicațiile nu sunt conectate);
- Prelucrarea paralelă pentru: replicare, cereri de regăsire, actualizare;
- Oferă facilități din tehnologia OO, prin care se permite definirea

şi utilizarea de obiecte mari şi complexe;

- Optimizează cererile de regăsire prin reutilizarea comenzilor SQL identice lansate de utilizatori diferiţi şi prin realizarea unui plan de execuţie a instrucţiunilor SQL;
- Are un grad de securitate sporit prin: server de criptare, control trafic reţea, niveluri de parolare etc.;
- Permite lucrul cu depozite de date (Data Warehouse) care conţin date multidimensionale (cu tehnologia OLAP);
- Conţine foarte multe produse ceea ce-l face să fie o platformă pentru baze de date: *servere* (Oracle 8, Application, Security, Internet Commerce etc), *instrumente* (Designer, Developer, Express, WebDB etc), *aplicaţii* (Financials, Projects, Market Manager, Manufacturing etc);
- Este primul SGBD pentru Internet cu server Java inclus;
- Reduce drastic costurilor pentru realizarea unei aplicaţii(de cca 10 ori faţă de versiunea anterioară);
- Este o platformă multiplă permiţând lucrul pe orice calculator, orice sistem de operare, orice aplicaţie, orice utilizator;
- Are instrumente diverse pentru dezvoltarea aplicaţiilor: bazate pe modelare (Designer, Developer, Application Server), bazate pe componente (Java), bazate pe HTML (browsere, editoare Web) şi XML, prin programare: proceduri stocate (PL/SQL, Java), obiecte standard, obiecte ODBC, obiecte JDBC, fraze SQL etc., tip internet (WebDB);
- Oferă servicii multiple de Internet (Web, E_mail, e_bussines, etc) integrate cu servicii Intranet.

Ulterior a fost lansat sistemul *Oracle 9i* care a marcat trecerea la o nouă generaţie de servicii internet. El este mai mult decât un suport pentru baze de date deoarece oferă o infrastructură completă de software pentru afaceri electronice (e-business) şi rulează pe o varietate de sisteme de calcul şi de operare: SUN-SOLARIS, HP-UX, IBM-AIX, PC_WINDOWS, XX-LINUX. Componenta Oracle WebDB a evoluat în Oracle Portal.

Oracle 9i DATABASE are faţă de versiunea anterioară asigură o protecţie ridicată şi automatizată iar costul administrării bazei de date scade în mod drastic.

Oracle 9i REAL APPLICATION CLUSTERS (RAC) se bazează pe o nouă arhitectură de BD numită *îmbinare ascunsă* (Cache Fusion). Aceasta este o nouă generaţie de tehnologie de clustere. Conform acestei arhitecturi la adăugarea unui calculator înr-o reţea cu BD Oracle, clusterele se adaptează automat la noile resurse, fără să fie necesară redistribuirea datelor

sau rescrierea aplicației. Posibilitatea apariției unei erori la o configurație cu 12 calculatoare sub Oracle 9i RAC este foarte mică, esimată ca durată în timp la cca 100.000 de ani.

În Oracle 9i APPLICATION SERVER se pot creea şi utiliza aplicații Web care sunt foarte rapide şi permit integrarea serviciilor de Internet.

Oracle 9i DEVELOPER SUITE este un mediu complet pentru dezvoltarea aplicațiilor tip afaceri electronice (e-business) şi tip Web. El se bazează pe tehnologiile Java şi XML şi permite personalizarea (Oracle Personalization).

În anul 2003 a fost lansată versiunea Oracle 10g care adaugă noi facilități sistemului Oracle 9i.

2.2. ARHITECTURA SISTEMULUI ORACLE

Componentele care formează arhitectura de bază Oracle (vezi fig.2.1) sunt dispuse într-o configurație client/server. Aceste componente sunt plasate pe calculatoare diferite într-o rețea asigurând funcționalități specifice, astfel: *serverul* asigură memorarea şi manipularea datelor, precum şi administrarea bazei de date iar *clientul* asigură interfața cu utilizatorul şi lansează aplicația care accesează datele din baza de date.

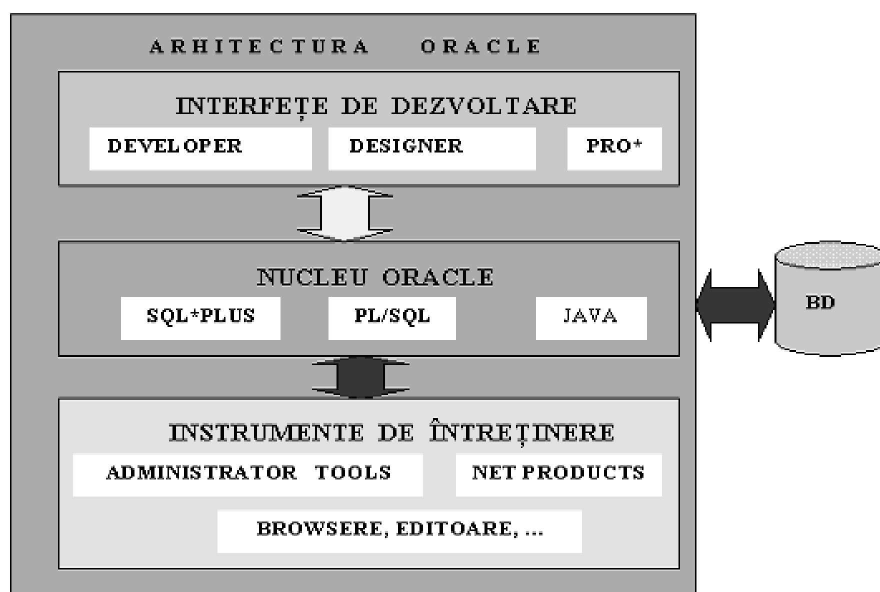


Figura 2.1 Arhitectura Oracle

Arhitectura Oracle se încadrează în tendințele actuale şi anume este structurată pe trei niveluri: nucleul, interfețele şi instrumentele de întreținere.

Nucleul Oracle conține componentele care dau tipul relațional pentru SGBD Oracle: limbajul relațional de regăsire SQL şi limbajul procedural propriu PL/SQL.

Sistemul Oracle creează şi întreține automat *dicționarul de date.* Acesta face parte din baza de date Oracle şi conține un set de tabele şi viziuni (vederi) accesibile utilizatorilor doar în consultare. Dicționarul conține informații de tipul: numele utilizatorilor autorizați, drepturile de acces, numele obiectelor din baza de date, structurile de date, spațiul ocupat de date, chei de acces etc.

Interfețele sunt componentele care permit dezvoltarea aplicațiilor cu BD, astfel:

- DEVELOPER SUITE este componenta destinată *dezvoltatorilor (programatorilor)* de aplicații. Conține generatoarele FORMS (meniuri şi videoformate), REPORTS (rapoarte şi grafice), JDEVELOPER;
- DESIGNER este componentă destinată *analiştilor/proiectanților* de aplicații. Oferă elemente de CASE pentru proiectarea aplicațiilor cu BD;
- PRO*C este componenta destinată *programatorilor în limbajele de programare universale* (FORTRAN, COBOL, Pascal, C, ADA, PL1);
- DATAWAREHOUSE BUILDER este destinat *analizei datelor multidimensionale*, folosind tehnologia de tip OLAP (On Line Analitical Processing);
- ORACLE APPLICATIONS permite dezvoltarea unor aplicații de întreprindere (Financials, Manufacturing, Projects etc.);

Instrumentele sunt componente destinate întreținerii şi bunei funcționări a unei BD Oracle. ENTERPRISE MANAGER CONSOLE conține mai multe utilitare destinate administratorului BD (deschidere/închidere BD, autorizarea accesului, refacerea BD, conversii de date, etc.).

2.3. ORACLE SERVER

Oracle Server (OS) permite managementul informațiilor organizate în baze de date, astfel încât se asigură accesul mai multor utilizatori în mod concurențial la aceleaşi date, oferind facilități de prevenire a accesului

neautorizat şi de restaurare a datelor după producerea unor erori. OS are următoarele facilităţi:

- Client/server permite ca prelucrările să fi împărţite între serverul de baze de date şi programele de aplicaţie ale utilizatorilor aflate pe staţiile conectate la server;
- Suportă lucrul cu baze de date foarte mari;
- Permite utilizarea concurenţială a bazelor de date;
- Oferă securitate sporită şi integritatea datelor;
- Permite lucrul distribuit;
- Conferă portabilitate aplicaţiilor;
- Permite ca mai multe tipuri de calculatoare şi sisteme de operare să coexiste pe aceeaşi reţea.

Oracle Server este un sistem relaţional-obiectual de management a bazelor de date, care permite o abordare deschisă, integrată şi cuprinzătoare a managementului informaţiilor.

OS constă dintr-un cuplu format dintr-o bază de date şi o instanţă Oracle.

A. **O *bază de date Oracle*** este o colecţie unitară de date, având o structură logică şi una fizică putând avea două stări: open (accesibilă) şi close (inaccesibilă).

1) Structura logică ale unei baze de date este formată din tabelele spaţiu (tablespaces), schema de obiectelor bazei de date, blocurile de date, extensiile şi segmentele.

Tabelele spaţiu sunt unităţile logice de memorie în care este împărţită o bază de date şi pot fi tabele spaţiu de sistem şi tabele spaţiu de utilizator. Din punct de vedere al accesibilităţii aceste pot fi on line şi off line.

Fişierele de date sunt structurile de memorie specifice unui sistem de operare pe care rezidă tabelele spaţiu ale unei baze de date.

Schema este o colecţie de obiecte, iar *schema de obiecte* este o structură logică ce se referă direct la datele unei baze de date(tabele, vederi, secvenţe, proceduri memorate, sinonime, indecşi, clustere şi link-uri de bază de date).

Blocurile de date, extensiile şi segmentele sunt elemente de control eficient al spaţiului de memorie externă pe disc aferent unei baze de date.

Blocul de date este unitatea de memorie cea mai mică manipulată de SGBD Oracle, iar mărimea acestuia măsurată în bytes se defineşte la momentul creerii bazei de date.

Extensia este format din mai multe blocuri de date contigue.

Segmentul este format din mai multe extensii. Segmentele pot fi: segmente de *date* (pentru memorarea datelor unei tabele), segmente de *indecşi*, segmente *roollback* (folosite pentru memorarea informaţiilor necesare pentru recuperarea datelor unei baze de date sau anularea unei tranzacţii) şi segmente *temporare* (folosite pentru prelucrarea instrucţiunilor SQL).

2) Structura fizică este definită de un set de fişiere specifice sistemului de operare pe care rezidă SGBD Oracle, folosite pentru memorarea structurilor logice ale bazei de date şi pentru păstrarea unor informaţii tehnice de control. Aceste fişiere sunt: fişiere de date (Data files), fişiere Redo log (Redo Log files) şi fişiere de control (Control files).

Fişierele de date (Data files) conţin datele unei baze de date, sub forma structurilor logice ale acesteia (tabele, vederi, secvenţe, proceduri memorate, sinonime, indecşi, clustere şi link-uri de bază de date). Fişierele de date au următoarele caracteristici: un fişier de date poate aparţine unei singure baze de date, pot fi extinse automat în anumite momente specifice ale funcţionării bazei de date, unul sau mai multe fişiere de date pot memora o tabelă spaţiu.

Fişierele Redo Log (Redo Log files) sunt folosite pentru memorarea tuturor schimbărilor de date produse asupra unei baze de date, astfel încât dacă se întâmplă o cădere de curent să se prevină distrugerea datelor bazei de date. Se pot folosi simultan mai multe fişiere de acest fel care să rezide pe discuri diferite.

Fişierele de control (Control files) sunt folosite pentru memorarea informaţiilor necesare pentru controlul structurii fizice a unei baze de date (numele bazei de date, numele şi locaţiile fişierelor de date, data creerii bazei de date etc).

B. ***Instanţa Oracle (Oracle instance)*** este combinaţia logică dintre structurile de memorie internă (SGA - system global area, PGA - program global area) şi procesele Oracle de bază activate la momentul pornirii unei baze de date.

1) SGA este o regiune partajabilă de memorie care conţine datele şi informaţiile necesare unei instanţe Oracle şi conţine:

- Database Buffer Cache (conţine blocurile de date cele mai recent utilizate pentru a reduce utilizarea discului);
- Redo Log Buffer (conţine datele despre blocurile modificate);
- Shared Pool (pentru prelucrarea instrucţiunilor SQL);
- Cursorii (Statement Handles or Cursores) folosiţi pentru manipularea instrucţiunilor unui limbaj gazdă folosind facilitatea Oracle Call Interface.

2) PGA este zona de memorie care conține datele şi informațiile de control ale unui proces server.

3) Procesul este un mecanism al sistemului de operare care poate executa o serie de paşi (instrucțiuni). Este cunoscut şi sub numele de job sau task. Procesul are propria sa zonă de memorie în care se execută. Un *server Oracle* are două tipuri de procese: *procese utilizator* şi *procese Oracle*.

Procesul utilizator (user proces) este creat şi menținut pentru a executa codul de program aferent unui anumit limbaj (C++) sau un produs Oracle (Oracle tool), SQL*Forms, Sql*Graphics etc.

Procesul Oracle este apelat de către un alt proces pentru a executa funcția cerută de către acesta. Procesele Oracle sunt Procese server şi procese background.

Procesele server (Server Processes) sunt utilizate de Oracle pentru a prelucra cererile proceselor utilizator. Oracle poate fi configurat astfel încât să permită unul sau mai multe procese utilizator. Din acest punct de vedere avem *servere dedicate* care au un singur proces utilizator şi *servere multi prelucrare* (multi-threaded server configuration). Pe anumite sisteme procesele utilizator şi procesele server sunt separate, iar în altele sunt combinate într-unul singur. Dacă folosim sistemul multi prelucrare sau dacă procesele utilizator şi procesele server se află pe maşini diferite atunci aceste procese trebuie să fie separate. *Sistemul client/server separă procesele utilizator de către procesele server şi le execută pe maşini diferite.*

Procesele background (Background processes) sunt create pentru fiecare instanță Oracle pentru a executa asincron anumite funcții. Acestea sunt:

- *Database Writer (DBWR)* scrie datele modificate în baza de date;
- *Log Writer (LGWR)* scrie înregistrările redo log pe disc;
- *Checkpoint (CKPT)* scrie înregistrările checkpoint la timpul potrivit ;
- *System Monitor (SMON)* execută recuperarea unei instanțe la momentul pornirii, colectează spațiul liber etc;
- *Process Monitor (PMON)* recuperează procesele utilizator dacă acestea cad accidental;
- *Archiver (ARCH)* copiază în mod online fişierele Redo Log în fişiere de arhivă atunci când acestea se umplu cu datei;
- *Recoverer (RECO)* rezolvă tranzacțiile suspendate în sistemul cu baze de date distribuite;
- *Dispacher (Dnnn)* folosit în sistemul multithreaded;

* *Lock (LCKn)* blochează procesele în sistemul Parallel server.

Legătura dintre procesele utilizator şi procesele Oracle este prezentată în figura 2.2.

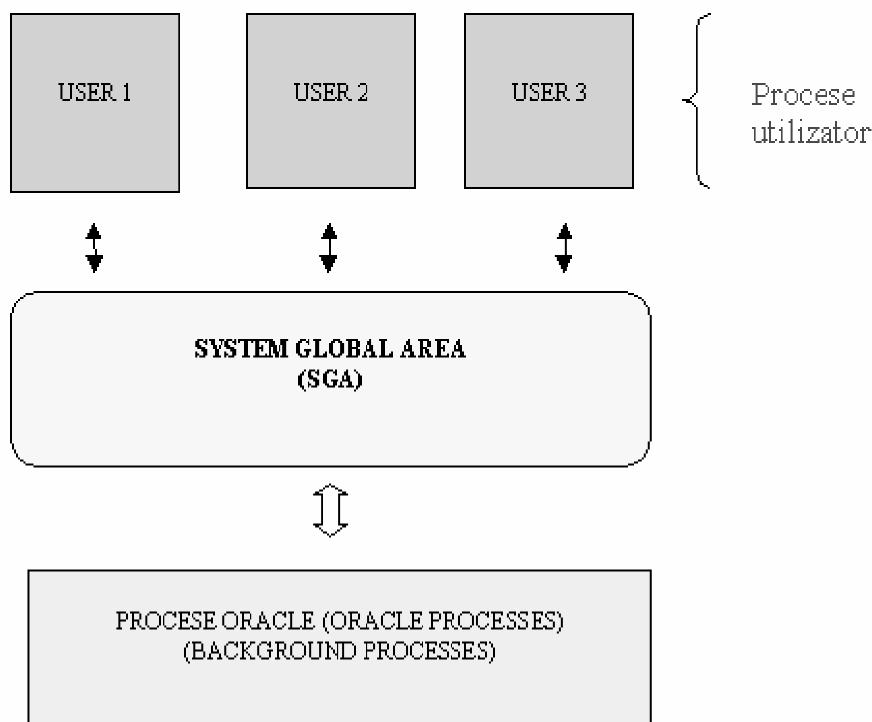


Figura 2.2 Legătura dintre procesele utilizator şi procesele Oracle

Interfaţa program este mecanismul de comunicare dintre un proces utilizator şi un proces server. Este metoda standard de comunicare între o aplicaţie sau un instrument Oracle şi Oracle Server.

2.4. CONCURENŢA, CONSISTENŢA ŞI SECURITATEA DATELOR

Într-un sistem de baze de date de tip multiutilizator o preocupare principală este asigurarea *accesului concurenţial* al mai multor utilizatori la aceleaşi date. Pentru această funcţie Oracle foloseşte diverse mecanisme ca blocarea înregistrărilor şi păstrarea mai multor versiuni consistente de date.

Consistenţa la citire garantează că setul de date văzut de către o instrucţiune nu se schimbă în timpul executării acesteia (consistenţă la nivel

de instrucțiune). Asigură faptul că un utilizator care accesează baza de date nu aşteaptă ca alt utilizator să scrie date sau să citească date şi că scrierea unor date în baza de date nu implică un timp de aşteptare pentru utilizatorii care doresc să citească aceste date. De asemenea, asigură faptul că un utilizator va aştepta la momentul scrierii în baza de date numai dacă încearcă să modifice acelaşi rând dintr-o tabelă (tranzacții concurente).

Mecanismul de blocare a rândurilor dintr-o tabelă a bazei de date asigură ca datele văzute de un utilizator sau modificate de acesta să nu poată fi modificate de către alt utilizator până când primul nu termină accesul la date. Datele şi structurile unei baze de date reflectă corect toate modificările efectuate într-o anumită secvență logică. Blocarea se poate executa automat sau manual.

Securitatea unei baze de date presupune asigurarea unor facilități care să permită controlul asupra modului în care o bază de date este accesată şi utilizată. Ea poate fi: *securitatea sistemului (*System security) şi *securitatea datelor* (Data security).

Securitatea sistemului include mecanisme care controlează accesul şi utilizarea bazei de date la nivel de sistem (validează combinațiile username/password, spațiul pe disc alocat pentru un anumit utilizator, limitele de resurse pentru un utilizator).

Securitatea datelor include mecanisme care controlează accesul şi utilizarea bazei de date la nivel de obiect (Exemplu utilizatorul SCOTT poate să emită instrucțiuni SELECT şi INSERT, dar nu poate utiliza DELETE).

Oracle Server furnizează *controlul accesului*, care înseamnă restricționarea accesului la informații pe bază de privilegii. De exemplu unui utilizator i se atribuie privilegiul de a accesa un anumit obiect al bazei de date. La rândul său acest utilizator, în mod corespunzător, poate să ofere privilegiul său altui utilizator.

Controlul securității în Oracle se asigură prin specificarea: utilizatorilor bazei de date, schemelor, privilegiilor, rolurilor, setarea limitelor de memorie, stabilirea limitelor de resurse şi auditarea.

Utilizatorii bazei de date şi schemele. Fiecare bază de date are o listă de nume de utilizatori. Pentru a accesa baza de date un utilizator trebuie să folosească o aplicație şi să se conecteze cu un nume potrivit. Fiecărui nume de utilizator îi este asociată o parolă. Orice utilizator are un domeniu de securitate care determină privilegiile şi rolurile, cota de tabelă spațiu alocată (spațiul pe disc alocat) şi limitele de resurse ce le poate utiliza (timp CPU etc).

Privilegiul este dreptul unui utilizator de a executa anumite instrucțiuni SQL. Privilegiile pot fi: privilegii de sistem şi privilegii de

obiecte. *Privilegiile de sistem* permit utilizatorilor să execute o gamă largă de instrucțiuni SQL, ce pot modifica datele sau structura bazei de date. Aceste privilegii se atribuie de obicei numai administratorilor bazei de date. *Privilegiile de obiecte* permit utilizatorilor să execute anumite instrucțiuni SQL numai în cadrul schemei sale, şi nu asupra întregii baze de date. *Acordarea privilegiilor* reprezintă modalitatea prin care acestea pot fi atribuite utilizatorilor. Există două căi de acordare e*xplicit* (privilegiile se atribuie în mod direct utilizatorilor) şi *implicit (*prin atribuirea acestora unor roluri, care la rândul lor sunt acordate utilizatorilor).

Rolurile sunt grupe de privilegii, care se atribuie utilizatorilor sau altor roluri. Rolurile permit:

- *Reducerea activităților de atribuire a privilegiilor.* Administratorul bazei de date în loc să atribuie fiecare privilegiu tuturor utilizatorilor va atribui aceste privilegii unui rol, care apoi va fi disponibil utilizatorilor;
- *Manipularea dinamică a privilegiilor.* Dacă se modifică un privilegiu de grup, acesta se va modifica în rolul grupului. Automat modificarea privilegiului se propagă la toți utilizatorii din grup;
- *Selectarea disponibilităților privilegiilor.* Privilegiile pot fi grupate pe mai multe roluri, care la rândul lor pot fi activate sau dezactivate în mod selectiv;
- *Proiectarea unor aplicații inteligente.* Se pot activa sau dezactiva anumite roluri funcție de utilizatorii care încearcă să utilizeze aplicația.

Un rol poate fi creat cu parolă pentru a preveni accesul neautorizat la o aplicație. Această tehnică permite utilizarea parolei la momentul pornirii aplicației, apoi utilizatorii pot folosi aplicația fără să mai cunoască parola.

Setarea cotelor de memorie ce pot fi folosite de către utilizatori se realizează folosind opțiunile:

- *Default tablespace*. Un utilizator poate crea obiecte ale bazei de date fără a specifica numele tabelei spațiu în care să fie create obiectele;
- *Temporary tablespace.* Unui utilizator i se alocă tabele spațiu proprii în care să-şi creeze obiectele;
- *Tablespace quotas.* Se pot seta limite fizice de memorie pentru tabelele spațiu proprii utilizatorilor.

Profilurile şi limitarea resurselor. Un profil este un element de securitate care permite manipularea resurselor ce pot fi alocate utilizatorilor. Resursele ce pot fi alocate sunt: numărul sesiunilor concurente, timpul CPU,

timpul de utilizare a unei sesiuni, restricții în utilizarea parolelor. Se pot crea diferite tipuri de profile care apoi vor fi atribuite fiecărui utilizator.

Auditarea permite monitorizarea activităților executate de către utilizatori astfel încât să se poată efectua investigații referitoare la utilizările suspecte ale bazei de date. Auditarea se poate efectua la nivel de *instrucțiune, privilegiu sau obiect*.

Recuperarea unei baze de date este necesară atunci când apar căderi de curent sau defecțiuni ale calculatorului. Tipurile de erori ce pot determina oprirea unei baze de date Oracle sunt: erori de utilizator; erori ale unor instrucțiuni sau ale proceselor utilizator; erori ale instanței Oracle; erori fizice pe disc.

Structurile fizice folosite de Oracle pentru recuperarea unei baze de date sunt *fişierele redo log*, *fişierele de control, segmentele rollback* şi *copiile fizice* ale datelor bazei de date.

Fişierele redo log permit protejarea datelor bazei de date actualizate în memoria internă dar nescrise încă în baza de date. Se pot utiliza în mod *online* sau cu *arhivare*. *Fişierele redo log on line* sunt un set de două sau mai multe fişiere care înregistrează toate tranzacțiile finalizate. Ori de câte ori o tranzacție este finalizată (comisă) datele modificate sunt scrise în aceste fişiere. Utilizarea fişierelor este ciclică, adică atunci când se umple un fişier se utilizează celălalt. *Fişierele redo log arhivate* permit arhivarea fişierelor redo log umplute înainte de a fi rescrise. Se poate rula în modul ARCHIVELOG (caz în care baza de date poate fi integral recuperată atât pentru o eroare a instanței, cât şi a discului) sau NOARCHIVELOG (caz în care baza de date poate fi recuperată numai după o eroare a instanței nu şi a discului). În primul mod recuperarea se face cu baza de date pornită, iar în al doilea caz cu ea oprită.

Fişierele de control conțin informații despre structura fişierelor bazei de date, numărul curent al secvenței de log folosit de către procesul LGWR etc.

Segmentele rollback se folosesc pentru controlul tranzacțiilor.

Copiile bazei de date pot fi integrale sau parțiale. *Copia integrală* cuprinde toate fişierele de date, online redo log files şi fişierele de control, iar *copia parțială* conține numai anumite părți ale bazei de date.

Datorită modului în care lucrează procesul DBWR fişierele de date ale bazei de date pot conține blocuri potențial actualizate de către tranzacțiile nefinalizate sau să nu conțină blocuri de date actualizate de către tranzacțiile finalizate. Blocurile de date conținând tranzacții finalizate nu au fost încă scrise în fişierele de date, ci numai în fişierele redo log, ceea ce înseamnă că fişierele redo log conțin modificări de date care trebuie efectuate şi în baza de date. Fişierele de redo log pot conține date ale unor

tranzacții nefinalizate care trebuie eliminate la momentul recuperării bazei de date.

Ca urmare a situațiilor de mai sus Oracle va folosi doi paşi distincți pentru recuperarea unei baze de date: *rolling forward* şi *rolling backward.*

Rolling forward înseamnă aplicarea (scrierea) asupra bazei de date a tuturor tranzacțiilor finalizate şi memorate în fişierele redo log. Se execută automat la momentul pornirii bazei de date dacă avem fişiere redo log online.

Rolling backward înseamnă ştergerea tuturor tranzacțiilor nefinalizate din fişierele redo log. Acest pas se execută automat după primul pas.

Utilitarul de recuperare Recovery Manager crează fişiere de salvare (backup) pentru fişierele de date ale bazei de date şi restaurează sau recuperează baza de date din acesrte fişiere backup.

2.5. DICȚIONARUL DE DATE (DATA DICTIONARY)

Fiecare bază de date Oracle are un dicționar de date, care este un set de tabele şi vederi folosite în modul read-only pentru a referi datele bazei de date. Dicționarul de date este actualizat automat de către Oracle ori de câte ori intervin modificări în structura bazei de date.

Dicționarul de date este alcătuit din *tabele de bază şi vederi* create pe aceste tabele. Tabelele de bază nu sunt accesibile datorită faptului că memorează datele criptic. Proprietarul dicționarului de date este utilizatorul SYS. Nici un utilizator nu poate altera obiecte din schema SYS.

Dicționarul de date (DD) este accesat în două situații: ori de câte ori Oracle prelucrează o instrucțiune DDL sau de către orice utilizator pentru consultarea informațiilor despre baza de date. DD este adus în memoria SGA. Este recomandat să nu se obiecte care să aparțină utilizatorului SYS. Nu se vor modifica niciodată date din DD. Singura tabelă care face excepție este tabela SYS.AUDIS. Această tabelă poate creşte mult în dimensiune, administratorul bazei de date putând şterge datele inutile.

Vederile DD sunt prefixate cu USER, ALL sau DBA. Vederile prefixate cu *USER* furnizează informații despre obiectele utilizatorilor, cele *ALL* despre toate obiectele din baza de date la care un utilizator are acces, iar cele cu *DBA* dau informații despre toată baza de date.

Exemple:

select object_name, object_type from user_objects;
select owner, object_name, object_type from all_objects;
select owner, object_name, object_type from sys.dba_objects;

Tabelele ce păstrează informații despre activitățile Oracle sunt tabele speciale care pot fi accesate numai de către administrator pentru a vedea performanțele Oracle. Utilizatorul SYS este proprietarul acestor tabele. Numele lor este prefixat cu *V_$,* iar sinonimele lor cu *V$*.
Categoriile de informații ce se pot obține din dicționarul de date sunt:

- Informații despre fişierele Online Redo Log;
- Informații despre tabelele spațiu;
- Informații despre fişierele de date (Data Files);
- Informații despre obiectele bazei de date;
- Informații despre segmentele bazei de date;
- Informații despre extensii ale bazei de date;
- Informații despre pachetele Oracle cu valoare de dicționar (Dictionary Storage).
- Informații despre utilizatorii bazei de date şi profilele acesteia;
- Informații despre privilegiile şi rolurile din baza de date

În tabelul 2.1 sunt descrise pachetele Oracle care permit PL/SQL să aibă acces la anumite facilități SQL sau să extindă funcționalitatea BD.

Pachete Oracle pentru accesul la facilitățile SQL

Tabelul 2.1.

DBMS_SPACE.UNUSED_SPACE	Returnează informații despre *spațiul nefolosit* dintr-un obiect (tabelă, index sau cluster)
DBMS_SPACE.FREE_BLOCKS	Returnează informații despre *blocurile libere* dintr-un obiect (tabelă, index sau cluster)
DBMS_SESSION.FREE_UNUSE D_ USER_MEMORY	Permite recuperarea memoriei nefolosite după efectuarea operațiilor care cer o cantitate mare de memorie (>100k)
DBMS_SYSTEM.SET_SQL_TRA CE_IN_SESSION	Permite sql_trace într-o sesiune identificată prin numărul serial şi SID (valori luate din V$SESSION).

2.6. ACCESUL LA DATE

Accesul la datele unei baze de date se realizează folosind instrucțiunile SQL (Structured Query Language) sau PL/SQL (Procedural Language).

Instrucțiunile SQL se împart în:

- Instrucțiuni de definire a datelor - DDL (Data Definition Statements). Acestea permit definirea, întreținerea şi ştergerea unor obiecte ale bazei de date;

- Instrucțiuni de manipulare a datelor – DML (Data Manipulation Statements), care permit regăsirea, inserarea, actualizarea şi ştergerea unor rânduri de date din tabele;
- Instrucțiuni de control a tranzacțiilor (Transaction Control Statements) permit controlul instrucțiunilor DML (COMMIT, ROLLBACK, SAVEPOINT etc);
- Instrucțiuni de control a sistemului Oracle (System Control Statements) permit utilizatorului să controleze proprietățile sesiunii curente prin activarea sau dezactivarea rolurilor sau setarea limbii;
- Instrucțiuni imprimate într-un limbaj gazdă (Embeded SQL Statements) şi încorporează instrucțiuni DDL, DML şi de control al tranzacțiilor.

O tranzacție este o unitate logică de lucru care cuprinde una sau mai multe instrucțiuni SQL executate de către un singur utilizator. Tranzacția începe cu prima instrucțiune SQL executabilă şi se termină în mod explicit cu *finalizarea* (commit) sau, după caz, *anularea tranzacției* (rollback).

Finalizarea unei tranzacții face ca modificările efectuate de intrucțiunilor SQL în baza de date să fie permanente, iar anularea (roll back) unei tranzacții duce la renunțarea la actualizările efectuate de instrucțiunile SQL până la un moment dat.

Tranzacțiile mari pot fi marcate cu puncte intermediare de salvare. Acest lucru permite ca activitățile efectuate între punctele de salvare să fie considerate finalizate, iar la momentul anulării (rollback) acest lucru să se execute până la un anumit punctul de salvare specificat.

PL/SQL este un limbaj procedural Oracle care combină instrucțiunile SQL cu instrucțiunile de control a prelucrării (IF … THEN, WHILE şi LOOP). Utilizarea procedurilor PL/SQL memorate în baza de date duce la reducerea traficului pe rețea. În baza de date pot fi stocate *proceduri, funcții, pachete, triggeri*.

Triggerii (declanşatorii) sunt blocuri de instrucțiuni scrise de programatori pentru a adăuga funcții suplimentare unei aplicații. Fiecare trigger are un nume şi conține una sau mai multe instrucțiuni PL/SQL. Un trigger poate fi asociat cu un eveniment şi poate fi executat şi întreținut ca un obiect distinct. Numele unui trigger corespunde unui eveniment (runtime events) care se produce la momentul execuției unei aplicații.

CAPITOLUL 3. ELEMENTELE DE BAZĂ ALE LIMBAJULUI SQL*PLUS

3.1. DESPRE LIMBAJ

În 1974 a fost lansat proiectul System/R de către firma IBM. Tot în acest an a apărut limbajul structurat de programare SEQUEL (Structured English as Query Language) autori fiind Chamberlin şi Boyce.

În 1976 apare un nou limbaj SEQUEL 2 care a fost declarat limbajul de interogare al SGBD System/R. Denumirea limbajului este schimbată de Chamberlin în SQL (Structured Query Language) în anul 1980.

Ulterior limbajul a fost perfecţionat fiind considerat cel mai răspândit limbaj de interogare a bazelor de date relaţionale.

Institutul Naţional pentru Standarde în anul 1982 a lansat un proiect de lucru pentru standardizarea limbajelor de interogare care a fost finalizat în 1986 apărând standardul ANSI SQL-86. Acesta defineşte comenzile de bază ale SQL, dar nu conţine partea de actualizare şi acordarea drepturilor de acces la o bază de date.

Prin revizuire acest limbaj apare în 1989 SQL-1 ca fiind limbajul fundamental al SGBD ralaţionale.

În 1992 apare versiunea SQL-2 care oferă noi facilităţi cum ar fi: joncţiune externă, implementarea restricţiei referenţiale, modificarea schemei bazei de date, etc.

Cel mai recent standard este SQL-3 care a fost lansat în anul 1999, acesta este considerat un limbaj complet în vederea definirii şi gestiunii obiectelor complexe. Se consideră că prin publicarea standardului propus în acest an a fost depăşită bariera relaţionalului, el fiind mult mai mult decât un instrument de consultare a bazelor de date.

3.2. CONCEPTE UTILIZATE

SQL*PLUS este un limbaj neprocedural şi operează asupra datelor normalizate. Conceptele necesare a fi cunoscute pentru lucrul cu acest limbaj sunt: tabelă, cheie primară, coloană, rînd, viziune, index, sinonim, cluster, bază de date relaţională, comanda, blocul, cererea,raportul etc.

Tabela sau relaţia este un ansamblu format din *n* coloane (atribute/subansambluri) şi *m* rânduri (tupluri/linii) care respectă următoarele condiţii minime: nu conţine date la nivel agregat (valorile aflate la intersecţia liniilor cu coloanele să fie la un nivel elementar); liniile sunt distincte unele faţă de altele; nu conţine coloane repetitive în descriere.

Cheia primară este un atribut care are valori distincte. Deci, fiecare linie se identifică printr-o valoare distinctă. Două sau mai multe atribute care pot fi chei primare se numesc chei candidate.

Coloana tabelei este formată din valorile pe care le ia atributul în liniile tabelei respective.

Rândul/tuplul/linia este format din valorile coloanelor ce se referă la o entitate a tabelei.

Baza de date relațională este un ansamblu de tabele normalizate, grupate în jurul unui subiect, în principiu, bine definit. Într-o bază de date relațională, entitățile şi legăturile sunt transpuse în tabele.

Viziunea este o tabela logică şi reprezintă o fereastră la date, dintr-una sau mai multe tabele.

Pentru ca accesul la date sa se facă mai rapid, se utilizează indexarea. Un *index* reprezintă o cheie pe una sau mai multe coloane. Indexarea este dinamică deoarece se pot adaugă sau şterge indecşi oricînd, fără ca datele memorate sau aplicațiile scrise să fie afectate.

Pentru realizarea unor operații sau pentru a utiliza în cereri nume mai scurte, se pot defini *sinonime* ale unor nume de tabele sau viziuni.

Un *cluster* reprezintă o anumită modalitate de grupare a rândurilor uneia sau mai multor tabele. Această grupare măreşte viteza de execuție a unor operații consumatoare de timp.

Comanda este o instrucțiune emisă din SQL*Plus către o bază de date Oracle.

Blocul reprezintă un grup de instrucțiuni SQL şi PL/SQL.

Cererea este o comanda SQL (SELECT) care regăseşte date din baza de date. Rezultatul cererii îl formează datele regăsite din baza de date.

Raportul este rezultatul cererii formatat cu ajutorul comenzilor SQL*Plus.

Numele unei baze de date, al unei tabele, coloane sau variabile utilizator trebuie să aibă lungimea între 1 şi 30 caractere. Un nume nu poate conține apostrofuri. Cu atît mai puțin, un nume utilizat într-o comandă nu va fi introdus între apostrofuri. Literele mici şi mari sunt echivalente (nu se face distincția între literele mici şi mari). Un nume trebuie să înceapă cu o literă, să conțină numai anumite caractere (A-Z, 0-9, $, #, @, -), să nu duplice numele unui alt obiect de acelaşi tip şi să difere de un cuvânt rezervat ORACLE.

Cuvintele rezervate nu pot fi utilizate ca nume de tabele, coloane sau orice alte obiecte definite de utilizator. Ele sunt prezentate în Anexa 1.

3.3. FUNCȚII SQL

Funcțiile se apelează prin sintaxa:

Nume_funcție (argument1, argument2, …)

Funcțiile SQL sunt c*u un singur rând* sau scalare (returnează un singur rezultat pentru fiecare rând al cererii emise asupra unei tabele sau vederi) şi *cu mai multe rânduri* numite funcții grup sau agregate (returnează un singur rezultat pentru un grup de rânduri regăsite dintr-o tabelă sau vedere).

A. Funcțiile SQL cu un singur rând (funcțiile scalare)

Acestea sunt funcții: numerice, caracter, de tip DATE, de conversie şi alte funcții.

1) Funcțiile numerice acceptă valori numerice şi returnează rezultate numerice şi sunt prezentate în tabelul 3.1. Tabela **DUAL** folosită în exemple**,** este creată automat de către Oracle odată cu crearea dicționarului de date şi se află în schema utilizatorului SYS, dar cu acest nume este accesibilă tuturor utilizatorilor unei baze de date Oracle. Ea are o singură coloană numită DUMMY cu tipul de date VARCHAR2(1) şi un singur rând cu valoarea 'X'. Selecțiile din tabela DUAL sunt utile pentru calcularea unor expresii constante cu comanda SQL SELECT. Din cauză că are un singur rând, constanta este returnată o singură dată. Alternativ pentru aceeaşi activitate se poate selecta o constantă, pseudocoloană sau o expresie din orice tabelă.

Tabelul 3.1 Funcțiile numerice

Funcția	***Rolul funcției***	***Exemple***
ABS(n)	*Returnează valoarea absolută a numărului n.*	*SELECT ABS(-15) "Absolut" FROM DUAL; Absolut ---------- 15*
ACOS(n)	*Arc cosinus de n. Rezultatul este exprimat în radiani.*	*SELECT COS(.3)"Arc_Cosinus" FROM DUAL; Arc cosinus ---------- 1.26610367*
ASIN(n)	*Arc sinus de n.*	*SELECT ASIN(.3) "Arc_Sinus" FROM DUAL; Arc_Sinus ---------- .304692654*
ATAN	*Arc tangentă de n.*	*SELECT ATAN(.3) "Arc_Tangentă" FROM DUAL;*

		Arc_Tangentă *----------* *.291456794*
ATAN2	*Arc tangentă de n şi m sau arc tangentă de n/m. Deci ATAN2(n,m) este identic cu ATAN2(n/m).*	*SELECT ATAN2(.3,.2) Arc_Tangentă2" FROM DUAL;* *Arc_Tangentă2* *------------* *.982793723*
CEIL(n)	*Returează numărul întreg cel mai mic care este mai mare sau egal cu n.*	*SELECT CEIL(15.7) "NUMĂR" FROM DUAL;* *NUMĂR* *----------* *16*
COS(n)	*Cosinus de n.*	*SELECT COS(180 * 3.14/180) "Cosinus de 180 grade" FROM DUAL;* *Cosinus de 180 grade* *---------------------* *-1*
COSH(n)	*Cosinus hiperbolic de n.*	*SELECT COSH(0) "Cosinus hiperbolic de 0" FROM DUAL;* *Cosinus hiperbolic de 0* *----------------------* *1*
EXP(n)	*Returnează o valoare egală cu e ridicat la puterea n, unde e = 2.71828183.*	*SELECT EXP(4) "e la puterea 4" FROM DUAL;* *e la puterea 4* *------------------* *54.59815*
FLOOR(n)	*Returnează numărul cel mai mare care este mai mic sau egal cu n.*	*SELECT FLOOR(15.7) "Floor" FROM DUAL;* *Floor* *----------* *15*
LN(n)	*Returnează logaritmul natural de n, unde n > 0.*	*SELECT LN(95) "Logaritm natural de 95" FROM DUAL;* *Logaritm natural de 95* *-----------------* *4.55387689*
LOG(m,n)	*Returnează logaritm în baza m de n.(*$LOG_m n$*)*	*SELECT LOG(10,100) "Log în baza 10 de 100" FROM DUAL;* *Log în baza 10 de 100* *------------------* *2*
MOD(m,n)	*Returnează restul împărţirii*	*SELECT MOD(11,4) "Modulo 4"*

	lui m la n. Dacă n este 0 returnează valoarea m. Această funcție se comportă diferit față de funcția modulo clasică din matematică, atunci când m este negativ. Având în vedere semnificația funcției MOD funcția modulo clasică din matematică se poate exprima cu formula : m - n * FLOOR(m/n)	FROM DUAL; Modulo 4 ---------- 3 2.
POWER (n,m)	Returnează o valoare egală cu m la puterea n.	SELECT POWER(3,2) "Putere" FROM DUAL; Putere ---------- 9
ROUND (n[,m])	Returnează n rotunjit la un număr de m yecimale. Dacă m este omis se elimină zecimalele, iar dacă este negativ se face rotunjirea numărului din dreapta virgulei zecimale, după regula: m = -1 rotunjire la nivel de zeci, m = -2 rotunjire la nivel de sute şi aşa mai departe.	SELECT ROUND(15.193,1) "Rotunjire" FROM DUAL; Rotunjire ---------- 15.2 SELECT ROUND(15.193,-1) "Rotunjire " FROM DUAL; Rotunjire ---------- 20 Aici rotunjirea s-a făcut la nivel de zeci, căci scala negativă înseamnă rotunjirea valorii din drepta semnului zecimal, iar scala pozitivă înseamnă rotunjirea numărului din dreapta semnului zecimal la ordinul de mărime cât este scala.
SIGN(n)	Returnează semnul numărului n, după regula: n<0 returnează valoarea -1; n=0 returnează valoarea 0; n>0 returnează valoarea +1.	SELECT SIGN(-15) "Semn" FROM DUAL; Semn ---------- -1
SIN(n)	Returnează sinus de n în radiani.	SELECT SIN(30 * 3.14159265359/180) "Sinus de 30 de grade" FROM DUAL; Sinus de 30 de grade ------------------ .5
SINH	Returnează sinus hiperbolic	SELECT SINH(1) " Sinus hiperbolic de

	de n.	*1"* *FROM DUAL;* *Sinus hiperbolic de 1* *--------------------* *1.17520119*
SQRT(n)	*Returnează rădăcină pătrată din n.*	*SELECT SQRT(26) "Rădăcină pătrată" FROM DUAL;* *Rădăcină pătrată* *-----------* *5.09901951*
TAN(n)	*Returnează tangentă de n.*	*SELECT TAN(135 * 3.14/180) "Tangentă de 135 grade"* *FROM DUAL;* *Tangentă de 135 grade* *----------------------* *- 1*
TANH(n)	*Returnează tangentă hiperbolică de n.*	*SELECT TANH(.5) " Tangentă hiperbolic de 5"* *FROM DUAL;* *Tangentă hiperbolic de 5* *------------------------* *.462117157*
TRUNC (n,m)	*Returnează valoarea lui n trunchiată la un număr de zecimale egal cu m. Dacă m este omis se elimină valorile zecimale, iar dacă ia o valoare negativă trunchiere se aplică părţii din stânga virgulei zecimale, după regula: m = -1 rotunjire la nivel de zeci, m= -2 rotunjire la nivel de sute şi aşa mai departe.*	*SELECT TRUNC(15.79,-1) "Trunc" FROM DUAL;* *Trunc* *----------* *10*

2) Funcţiile caracter acceptă la intrare valori caracter şi furnizează valori caracter sau numerice. În tabelul 3.2 sunt prezentate funcţiile caracter care *returnează caractere*, iar în tabelul 3.3 funcţiile caracter care *returnează valori numerice.* Valorile caracter returnate sunt de tipul VARCHAR2 dacă nu se specifică altfel.

Tabelul 3.2 Funcţiile caracter care returnează caractere

Funcţia	***Rolul funcţiei***	***Exemple***
CHR(n)	*Returnează caracterul care are valoarea binară n.*	*SELECT CHR(67)||CHR(65)||CHR(84) "Caractere"* *FROM DUAL;* *Caractere*

		--- CAT
CONCAT (c1,c2)	*Returnează o valoare formată din concatenarea caracterului c1 cu caracterul c2.*	*SELECT CONCAT(CONCAT (nume, ' este '), funcție) "Funcție" FROM tab1 WHERE codfuncție = 7000; Funcție ----------------- Popescu este PROGRAMATOR*
INITCAP ('şir')	*Returnează şirul de caractere 'şir' astfel încât fiecare cuvânt al şirului are prima literă în format literă mare.*	*SELECT INITCAP('cuvânt1 cuvânt2') "Litere mari" FROM DUAL; Litere mari --------- Cuvânt1 Cuvânt2*
LOWER ('şir')	*Returnează şirul de caractere 'şir' astfel încât toate literele şirului sunt de format literă mică.*	*SELECT LOWER('BUCUREŞTI') "Literă mică" FROM DUAL; Literă mică. -------------------- bucureşti*
LPAD ('c1',n,'c2')	*Returnează şirul de caractere 'c1', pe lungime de n caractere, astfel încât partea din stânga şirului până la lungimea de n caractere este umplută cu secvențe de caractere egale cu 'c2'. Dacă şirul 'c1' este mai lung decât valoarea n, atunci se returnează partea dreaptă a şirului pe lungime de n caractere.*	*SELECT LPAD('Page.1',10,'*.') "LPAD exemplu" FROM DUAL; LPAD exemplu --------------- *.*.*Pag.1*
LTRIM ('c1' [,'c2'])	*Returnează partea din şirul 'c1' care a mai rămas după ce au fost înlăturate toate caracterele din stânga acestuia care se regăsesc în setul de caractere 'c2'.*	*SELECT LTRIM('xyxXxyREST ŞIR','xy') "LTRIM exemplu" FROM DUAL; LTRIM exemplu ------------ XxyREST ŞIR*
NLSSORT ('c1' [,'nlsparams'])	*Returnează un şir de caractere folosite pentru sortarea câmpurilor de tip caracter. Caracterul 'c1' defineşte un marcator de sortare, în sensul că toate valorile dintr-un câmp care sunt mai mari sau mai mici decât acesta vor fi afişate sau nu prin folosirea*	*SELECT nume FROM tab1 WHERE NLSSORT(nume,'NLS_SORT=romanian')< NLSSORT('O','NLS_SORT=romanian') ORDER BY nume; NUME ----------*

	clauzei ORDER BY. Parametrul 'nlsparams' defineste valoare lingvistică după care să se facă sortare şi se dă sub forma 'NLS_SORT = sort' în care sort defineşte limba după care dorim să se facă sortarea(germană, română etc).	*IONESCU* *MARINESCU* *POPESCU* *OLGA* *Notă: Numele care încep cu o literă mai mare decât 'O' nu se vor afişa.*
REPLACE ('c1','c2','c3')	*Returnează şirul 'c1' translatat, astfel încât în şirul 'c1' toate valorile egale cu şirul de căutare 'c2' sunt înlocuite cu şirul de înlocuire 'c3'.*	*SELECT REPLACE ('MASĂ şi MUSCA','M','C') "REPLACE" FROM DUAL;* *REPLACE* -------------- *CASĂ şi CUŞCĂ*
RPAD ('c1',n[,'c2'])	*Returnează şirul de caractere 'c1', pe lungime de n caractere, astfel încât partea din dreapta şirului până la lungimea de n caractere este umplută cu secvenţe de caractere egale cu 'c2'. Dacă şirul 'c1' este mai lung decât valoarea n, atunci se returnează partea stângă a şirului pe lungime de n caractere.*	*SELECT RPAD('CLUJ',10,'ab') "RPAD exemplu"* *FROM DUAL;* *RPAD exemplu* ----------------- *CLUJababab*
RTRIM ('c1','c2')	*Returnează partea din şirul 'c1' care a mai rămas după ce au fost înlăturate toate caracterele din dreapta acestuia care se regăsesc în setul de caractere 'c2'.*	*SELECT RTRIM('BUCUREŞTIyxXxy','xy') "RTRIM exemplu"* *FROM DUAL;* *RTRIM exemplu* ------------- *BUCUREŞTIyxX*
SUBSTR ('c1',m[,n])	*Returnează porţiunea din şirul 'c1' care începe de la al m-lea caracter pe lungime de n caractere.*	*SELECT SUBSTR ('ABCDEFG',3.1,4) Subşir1"* *FROM DUAL;* *Subşir1* ---- *CDEF* *SELECT SUBSTR('ABCDEFG',-5,4) "Subşir2"* *FROM DUAL;* *Subşir2* ---- *CDEF*

TRANSLATE ('c1','c2','c3')	*1. Translatează şirul 'c1' prin intermediul şirului 'c2' la valorile din şirul 'c3' după regula: fiecare caracter din şirul c1 este căutat în şirul 'c2', dacă este găsit valoarea acestuia este înlocuită cu caracterul din şirul 'c3' a cărui poziţie corespunde cu poziţia caracterului din şirul 'c2'.* *2. Dacă şirul 'c2' este mai lung decât şirul 'c3' caracterele ce nu pot fi translatate sunt eliminate din şirul 'c1'.*	*SELECT TRANSLATE ('2KRB229', '0123456789ABCDEFGHIJKLMNOPQ RSTUV', '9999999999XXXXXXXXXXXXXXXXXX XXXXXXXX') "TRANSLATE 1" FROM DUAL;* *TRANSLATE 1* *--------* *9XXX999* *SELECT TRANSLATE ('2KRW229', '0123456789ABCDEFGHIJKLMNOPQ RSTUVWX', '0123456789') " TRANSLATE 2" FROM DUAL;* *TRANSLATE 2* *-----------------* *2229*
UPPER('c1')	*Returnează şirul 'c1' cu toate caracterele transformate în caractere mari.*	*SELECT UPPER('Bucureşti') "LITERE MARI" FROM DUAL;* *LITERE MARI* *-----* *BUCUREŞTI*

Tabelul 3.3 Funcţiile caracter care returnează valori numerice

Funcţia	***Rolul funcţiei***	***Exemple***
ASCII ('c1')	*Returnează valoarea zecimală a primului caracter din şirul 'c1'.*	*SELECT ASCII('Q') FROM DUAL;* *ASCII('Q')* *----------* *81*
INSTR ('c1','c2' [,n[,m]])	*1.Caută în şirul 'c1' începând cu al n-lea său caracter a m-a apariţie a şirului 'c2' şi returnează poziţia acestuia în şirul 'c1' relativ la începutul şirului. Dacă şirul 'c2' nu este găsit în şirul 'c1' se returnează valoarea 0. Valorile asumate prin lipsă pentru n şi m sunt 1.* *2. Dacă n este negativ căutarea se face invers de la sfârşitul şirului.*	*SELECT INSTR ('CORP**OR**ATE FLO**OR**','OR', 3, 2) "INSTR" FROM DUAL;* *INSTR* *----------* *14* *SELECT INSTR ('CORPORATE FLOOR', 'OR', -3, 2) "INSTR INVERS" FROM DUAL;* *INSTR INVERS* *-----------------* *2*
LENGTH	*Returnează lungimea în*	*SELECT LENGTH('BUCUREŞTI')*

	caractere a şirului de caractere 'c1' de tip CHAR.	*"LUNGIME ŞIR"* *FROM DUAL;* *LUNGIME ŞIR* *--------------------* *9*

3) Funcțiile de tip DATE operează cu tipuri de date de tip DATE şi sunt prezentate în tabelul 3.4. Toate funcțiile de tip DATE returnează valori de tip DATE, mai puțin funcția MONTH_BETWEEN care furnizează o valoare numerică. Structurile formatului *fmt* de afişare a datelor pentru funcțiile de tip DATE sunt prezentate în tabelul 3.7.

Tabelul 3.4 Funcțiile de tip DATE

Funcția	***Rolul funcției***	***Exemple***
ADD_MONTHS(d,n)	*Returnează data d plus un număr de luni egal cu n.*	*Dacă în coloana data1 aferentă numelui 'POPESCU' din tabela tab1 avem data 17 septembrie2005 cu comanda de mai jos se va ob'ine data 17 octombrie 2005.* *SELECT TO_CHAR (**ADD_MONTHS**(data1,1), 'DD-MON-YYYY') "Luna următoare"* *FROM tab1* *WHERE nume = 'POPESCU';* *Luna următoare* *-----------* *17-OCT-2005*
LAST_DAY (d)	*Returnează data ultimei zile din lună.*	*Cu această funcție se poate determina numărul zilelor rămase din luna curentă.* *SELECT SYSDATE, **LAST_DAY(SYSDATE)** "ULTIMA",LAST_DAY(SYSDATE) - SYSDATE "ZILE RĂMASE"* *FROM DUAL;* *SYSDATE ULTIMA ZILE RĂMASE* *--------- --------- ----------* *23-SEP-05 30-SEP-05 7* *SELECT TO_CHAR(ADD_MONTHS(LAST_DAY(data1), 5), 'DD-MON-YYYY') "Cinci luni"* *FROM tab1* *WHERE nume = 'POPESCU';* *Cinci luni* *-----------* *28-FEB-2006*
MONTHS_ BETWEEN (d1, d2)	*Returnează numărul de luni dintre datele d1*	*SELECT MONTHS_BETWEEN (TO_DATE('02-09-2005','MM-DD-YYYY'), TO_DATE('01-08-2005','MM-DD-YYYY'))*

	şi d2. Dacă d1 şi d2 sunt aceleşi zile din lună sau sunt ultimele zile din lună rezultatul este un număr întreg, altfel Oracle calculează fracţiuni din lună bazat pe o lună cu 31 zile.	*"Luni" FROM DUAL; Luni ---------- 1.03225806*
NEXT_DAY (d, 'c1')	*Returnează data primei zile a săptămânii după ziua definită de şirul 'c1' şi care este după data d.*	*În exemplul de mai jos se returnează data zilei care urmează zilei de Marţi, după data de 15 martie 1999. SELECT NEXT_DAY('15-MAR-05','TUESDAY') "ZIUA URMĂTOARE" FROM DUAL; ZIUA URMĂTOARE --------- 22-MAR-05*
ROUND (d[,fmt])	*Returnează data d rotunjită la unitatea de timp specificată de către formatul* ***fmt,*** *conform specificaţiilor de la sfârşitul tabelului.*	*SELECT ROUND (TO_DATE ('27-SEP-05'),'YEAR') "Noul an" FROM DUAL; Noul an --------- 01-JAN-06*
SYSDATE	*Returnează data şi timpul curent.*	*SELECT TO_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS') "Data şi timpul curent" FROM DUAL; Data şi timpul curent ------------------- 27-09-2005 20:27:11*
TRUNC (d,[fmt])	*Returnează data d fără timp trunchiată la o unitate specificată de formatul fmt, iar dacă este absent se face trunchierea la ziua cea mai*	*SELECT TRUNC(TO_DATE ('27-SEP-05','DD-MON-YY'), 'YEAR') "Anul nou" FROM DUAL; Anul nou --------- 01-JAN-05*

	apropiată.	

*Formatul **fmt** utilizat de funcțiile **ROUND şi TRUNC***	
Formatul fmt	***Semnificația formatului fmt***
CC, SCC	*Se rotunjeşte la nivel de secol (primii doi digiți ai anului exprimat pe patru digiți + 1)* *Exemplu: **18**98 se rotujeşte la **19**98.*
SYYYY, YYYY, YEAR SYEAR, YYY, YY, Y	*Se rotunjeşte la nivelul 01 ianuarie a anului din data care se rotunjeşte.* *Exemplu: 27-sep-05 se rotunje;te la 01-jan-05.*
Q	*Rotunjire la nivel de trimestru (Se routunjeşte în sus la a şasesprezecea zi a lunii a doua a trimestrului).*
MONTH, MON, MM, RM	*Rotunjire la nivel de lună (Se routunjeşte în sus la a şasesprezecea zi a lunii).*

4) Funcțiile de conversie convertesc o valoare de la un tip de dată la alt tip de dată. Aceste funcții au formatul general de forma ***tip_de_dată TO tip_de_dată*** şi sunt prezentate în tabelul 3.5.

Tabelul 3.5 Funcțiile de conversie

Funcția	***Semnificația***	***Exemple***
CONVERT ('c1', set_destinație, [,set_sursă])	*Converteşte un şir de caractere de la un set de caractere la alt set de caractere. Setul set_sursă este setul de caractere din care fac parte caracterele şirului 'c1', iar set_destinație este setul de caractere în care se convertesc caracterele şirului 'c1'.*	*SELECT CONVERT('Groß', 'US7ASCII', 'WE8HP') "Conversie" FROM DUAL;* *Conversie* *----------* *Gross* *Seturile de caractere cele mai comune sunt: US7ASCII, WE8DEC, WE8HP, F7DEC, WE8EBCDIC500 , WE8PC850, WE8ISO8859P1 .*
HEXTOROW('c1')	*Converteşte şirul 'c1' care conține digiți hexazecimali la tipul de date RAW.*	*INSERT INTO tab1 (raw_column) SELECT HEXTORAW('7D') FROM DUAL;*
RAWTOHEX(raw)	*Converteşte digiți hexazecimali de tip RAW la şirul 'c1'.*	
ROWIDTOCHAR (rowid)	*Converteşte valoarea ROWID la o valoare de tip VARCHAR". Rezultatul conversiei*	*SELECT ROWID* *FROM tab1* *WHERE ROWIDTOCHAR(ROWID) LIKE '%Br1AAB%';*

	este tot impul un şir de 18 caractere.	*ROWID* *------------------* *AAAAZ6AABAAABr1AAB*
TO_CHAR pentru conversie de caractere, are sintaxa: *TO_CHAR* *(d [, fmt])*	*Converteşte data d de tip DATE la o valoare de tip VARCHAR2 în formatul specificat.*	*SELECT TO_CHAR (data1, 'Month DD, YYYY') "Format nou"* *FROM tab1* *WHERE nume = 'ION';* *Format nou* *------------------* *May 01, 2005*
TO_CHAR pentru conversie de numere, are sintaxa: *TO_CHAR* *(n [, fmt])*	*Converteşte numărul n de tip NUMBER la o valoare de tip VARCHAR2.*	*SELECT TO_CHAR(-10000,'L99G999D99MI') "Cantitate"* *FROM DUAL;* *Cantitate* *--------------* *$10,000.00-*
CHARTOROWID *('c1')*	*Converteşte o valoare de tip CHAR sau VARCHAR2 la o valoare de tip ROWID*	*SELECT nume* *FROM tab1* *WHERE ROWID = CHARTOROWID ('AAAAfZAABAAACp8AAO');* *nume* *----------* *POPESCU*
TO_DATE *('c1' [, fmt])*	*Converteşte şirul de caractere 'c1' de tip CHAR sau VARCHAR2 la o valoare de tip DATE în conformitate cu formatul fmt . Formatul fmt pentru datele de tip DATE este prezentat în tabelul 3.7.*	*INSERT INTO tab1 (data1)* *SELECT TO_DATE('January 15, 2005, 11:00 A.M.', 'Month dd, YYYY, HH:MI A.M.')* *FROM DUAL;*
TO_NUMBER *('c1',[,fmt])*	*Converteşte şirul de caractere 'c1' de tip CHAR sau VARCHAR2 la o valoare numerică de tip NUMBER în conformitate cu formatul fmt. Forma acestui format este în tabelul 3.6.*	*UPDATE tab1* *SET salariu = salariu +* *TO_NUMBER('100.00', '9G999D99')* *WHERE nume = 'ION';*

Tabelul 3.6 Structurile formatului **fmt** pentru datele de tip NUMBER

Elementul fmt	***Exemple***	***Semnificația elementului fmt***
9	*9999*	*Nr. semnificativ de digiți care vor fi afişați*
0	*0999 9990*	*Afişează 0 în fața sau după digiții semnificativi*

$	*$999*	*Afişează semnul $ în faţa numărului*
B	*B999*	*Afişează valorile 0 ca blank-uri*
MI	*999MI*	*Afişează semnul "-" după numerele negative*
S	*S999*	*Afişează semnul "+" sau "-" în faţa numerelor pozitive, respectiv negative*
D	*99D99*	*Afişează punctul zecimal în această poziţie*
G	*99G999*	*Separator de grupuri*
,	*99,999,99*	*Virgula se afişează în poziţiile indicate*
.	*999.99*	*Afişează punctul zecimal în această poziţie*
RN sau rn	*RN sau rn*	*Afişează cifre romane cu majuscule, respectiv cu caractere mici(minuscule)*

Tabelul 3.7 Structurile formatului fmt pentru datele de tip DATE

Elementul fmt	***Se specifică în TO-DATE***	***Semnificaţia elementului fmt***
- / , . ; :	*Da*	*Punctuaţii pentru dată*
"text"	*Da*	*Text reprodus în rezultat*
AD sau A.D.	*Da*	*Specificarea unui an din Era Noastră(E.N.)*
BC sau B.C.	*Da*	*Specificarea unui an Înaintea Erei Noastre(Î.E.N.)*
CC sau SCC	*Nu*	*Secolul = cu primii doi digiţi ai anului pe patru digiţi + 1*
D	*Da*	*Ziua din săptămână(de la 1 la 7)*
DAY	*Da*	*Numele zilei*
DD	*Da*	*Ziua din lună(de la 1 la 31)*
DDD	*Da*	*Ziua din an(1 la 366)*
HH sau HH12	*Da*	*Ora din zi din intervalul 1 - 12*
HH24	*Da*	*Ora din zi din intervalul 1 - 24*
J	*Da*	*Ziua din calendarul Iulian cu valori începând cu 1 ianuarie 4712 BC. Trebuie să fi întreg.*
MI	*Da*	*Minute (0 la 59)*
MM	*Da*	*Luna (01 la 12)*
MON	*Da*	*Numele prescurtat al lunii*
MONTH	*Da*	*Numele întreg al lunii*
PM sau P.M.	*Nu*	*Indicator de meridian*
RR sau RRRR	*Da*	*Rotunjirea anului pe doi digiţi sau patru digiţi*
SS	*da*	*Secundele în cadrul minutului(0 la 59)*
SSSS	*Da*	*Secundele de la miezul nopţii în cadrul zilei(0 la 86399)*
WW	*Nu*	*Săptămâna din an (1 la 53)*
W	*Nu*	*Săptămâna în cadrul lunii(1 la 5)*
YYYY,YYY,YY,Y	*Da*	*Anul cu 4 , 3, 2, 1 digiţi.*

Alte funcţii cu un singur rând sunt prezentate în tabelul 3.8.

Tabelul 3.8 Alte funcții cu un singur rând

Funcția	*Rolul funcției*	*Exemple*
DUMP ('c1' [,return_format [,start_position [,length]]])	*Returnează o valoare de tip VARCHAR2 conținând codul tipului de date, lungimea în baiți şi reprezentarea internă a expresiei expr return_format este codul sistemului de numerație în care este reprezentată valoarea returnată de funcție pentru şirul 'c1'; Dacă se furnizează ca o valoare egală cu codul sistemului de numerație + 1000 se returnează şi numele sistemului de numerație. start_position determină poziția din şirul 'c1' de unde să înceapă DUMP-u length este lungimea DUMP-ului.*	*SELECT DUMP('abc', 1016) FROM DUAL; DUMP('ABC',1016) ---------------------------------- Typ=96 Len=3 CharacterSet=WE8DEC: 61,62,63 SELECT DUMP(nume, 8, 3, 2) "OCTAL" FROM tab1 WHERE nume = 'SCOTT'; OCTAL ---------------------------- Type=1 Len=5: 117,124 Sistemele de numerație sunt: 8 = sistemul octal; 10 = sistemul zecimal; 16 = sistemul hexazecimal; 17 = rezultatul este returnat sub forma de caractere singulare.*
EMPTY_ [B|C]LOB()	*Returnează un pointer sau locator care poate fi folosit pentru inițializare unei variabile de tip LOB, într-un INSERT, într-un UPDATE pentru inițializarea unei coloane de tip LOB sau ca atribut EMPTY, care înseamnă că LOB-ul a fost inițializat dar nu a fost populat cu date.*	*INSERT INTO lob_tab1 VALUES (EMPTY_BLOB()); UPDATE lob_tab1 SET clob_col = EMPTY_BLOB();*
BFILENAME ('director', 'nume_fişier')	*Returnează un pointer sau locator asociat cu un LOB de tipul unui fişier binar de pe server. director este numele directorului unde se află fişierul LOB de tip binar. nume_fişier este numele fişierului.*	*INSERT INTO tab1 VALUES (BFILENAME('lob_dir1',foto1.gif')) ;*
GREATEST (expr [,expr] ...)	*Returnează valoarea cea mai mare dint-o listă de valori.*	*SELECT GREATEST ('HARRY', 'HARRIOT', 'HAROLD') "Mare" FROM DUAL; Mare -----*

		HARRY
LEAST (expr [,expr] ...)	*Returnează valoarea cea mai mică dint-o listă de valori.*	*SELECT LEAST('HARRY','HARRIOT','HAROLD') "Mică" FROM DUAL;* *Mică* *------* *HAROLD*
NVL (expr1, expr2)	*Dacă expr1 este NULL returnează expresia expr2, iar dacă expr1 nu este NULL returnează expr1. Argumentele expr1 şi expr2 pot avea orice tip de date. Dacă sunt de tipuri diferite Oracle converteşte expr2 la tipul expr1 înainte de a executa comparaţiile. Valoarea returnată este totdeauna de tipul expr1, execpţie făcând situaţia când expr1 este de tip caracter, caz în care rezultatul este VARCHAR2.*	*SELECT nume, NVL(TO_CHAR(comision), 'NOT APPLICABLE') "COMISION" FROM tab1 WHERE codepart = 30;* *NUME COMISION* *---------- ------------------* *ALLEN 300* *WARD 500* *MARTIN 1400* *TURNER 0* *JAMES NOT APPLICABLE*
UID	*Returneză un întreg care identifică în mod unic utilizatorul curent.*	
USER	*Returnează identificatorul utilizatorului curent în format VARCHAR2.*	*SELECT USER, UID FROM DUAL;* *USER UID* *----------------------------* *SCOTT 19*
USERENV (option)	*Returnează informaţii despre sesiune curentă.*	*SELECT USERENV('LANGUAGE') "Limbajul" FROM DUAL;* *Limbajul* *-----------------------------------* *AMERICAN_AMERICA.WE8DEC*
VSIZE(expr)	*Returnează lungimea în baiţi a expresiei expr.*	*SELECT nume, VSIZE (nume) "BYTES" FROM tab1 WHERE codepart = 10;* *NUME BYTES* *---------------------------* *CLARK 5* *KING 4* *MILLER 6*

B.Funcțiile cu mai multe randuri (de grup)

Furnizează un rezultat bazat pe prelucrarea mai multor rânduri. Toate funcțiile de grup, mai puțin COUNT(*) ignoră valorile NULL.

Majoritatea funcțiilor de grup acceptă opțiunile:**DISTINCT** (determină luarea în calcul numai a valorilor distincte ale rândurilor din cadrul grupului) şi **ALL** (determină luarea în calcul a tuturor valorilor grupului de rânduri).

Funcțiile de grup sunt prezentate în tabelul 3.9.

Tabelul 3.9 Funcțiile de grup

Funcția	***Semnificația***	***Exemple***
AVG([DISTINCT|ALL] n)	*Returneză media celor n valori*	*SELECT AVG(salariu) "Medie" FROM tab1;* *Media* *----------* *2077343.21429*
COUNT ({ | [DISTINCT|ALL] expr})*	*Returnează toate rândurile cererii. Dacă avem argumentul = * se returnează toate rândurile indiferent de valoarea lor (NULL sau NOT NULL)*	*SELECT COUNT(*) "Total" FROM tab1;* *Total* *----------* *18* *SELECT COUNT(job) "Count" FROM tab1;* *Count* *----------* *14* *SELECT COUNT(DISTINCT job) "Jobs" FROM emp;* *Jobs* *----------* *5*
MAX([DISTINCT|ALL] expr)	*Returnează maximul din expresia expr.*	*SELECT MAX(salariu) "Maximum" FROM tab1;* *Maximum* *----------* *5000*
MIN([DISTINCT|ALL] expr)		*SELECT MIN(data1) "Minim" FROM tab1;* *Minimum*

		--------- *17-DEC-80*
SUM([DISTINCT|ALL] n)		*SELECT SUM(salariu) "Total" FROM tab1;* *Total* *----------* *29081*

Utilizatorii pot să-şi scrie propriile funcţii în PL/SQL pentru a executa activităţi neacoperite de către funcţiile SQL. Aceste funcţii pot fi folosite în comenzile SQL la fel ca şi cele standard.
De exemplu, funcţiile utilizator **pot fi folosite** în: lista de selecţie a comenzii SELECT; condiţia din clauza WHERE; clauzele CONNECT BY, START WITH, ORDER BY ŞI GROUP BY ; clauza VALUES a comenzii INSERT; clauza SET a comenzii UPDATE.

Funcţiile utilizator **nu pot** fi folosite în clauzele CONSTRAINT sau DEFAULT ale comenzilor C REATE TABLE sau ALTER TABLE şi nici pentru actualizarea bazei de date. În funcţiile utilizator nu este permisă utilizarea parametrilor OUT sau IN OUT.

3.4. EXPRESII SQL

Expresia este o combinaţie de unul sau mai mulţi operatori, operanzi (variabile, literali, coloane, funcţii SQL) care se evaluează la o singură valoare.

Operatorii utilizaţi în comenzile SQL sunt: operatori SQL*PLUS; operatori SQL; operatori aritmetici; operatori logici; operatori specifici în expresiile de cereri. Ei sunt prezentaţi în Anexa 2, în ordinea descrescătoare a priorităţii, cei de aceeaşi importanţă fiind grupaţi între perechi de linii orizontale. Operatorii sunt evaluaţi de la stînga spre dreapta.

O expresie are în general acelaşi tip de date ca şi componentele sale. Expresiile sunt folosite pentru construirea instrucţiunilor SQL şi a unor liste de expresii.

A. Există cinci forme de furnizarea a expresiilor pentru **construirea instrucţiunilor SQL**.

1) Forma I este formată din coloană, pseudocoloană, constantă, secvenţă sau NULL şi are sintaxa:

nume_schemă.tabelă | vedere. coloană | pseudocoloană | ROWLABEL
sau
text | număr | nume_secvenţă | nume secvenţă. CURRVAL | NEXTVAL | NULL

Pentru *nume_schemă* se poate folosi pe lângă numele schemei utilizatorului şi valoarea "PUBLIC", care califică sinonimele publice pentru o tabelă sau o vedere, calificare care este suportată doar în instrucțiunile SQL pentru manipularea datelor de tip (DDL), nu şi în cele de definire a datelor de tip DDL.

Pseudocoloană poate fi doar LEVEL, ROWID sau ROWNUM.

Exemple:

Tab1.numecol_1
'acesta este un şir de caractere'
10
secventa1.CURRVAL

2) Forma II este folosită pentru definirea unei variabile gazdă (host variable) cu sau fără indicator de variabilă. Expresiile din această formă se vor folosi doar în instrucțiunile SQL incluse în limbajele de programere gazdă sau în instrucțiunile prelucrate de programele OCI(Oracle Call Interface). Sintaxa este de forma:

: variabilă_gazdă INDICATOR :variabilă_indicator

Exemple:

: nume_angajat INDICATOR :indicator_var_nume_angajat
:nume_persoană

3) Forma III este folosită pentru apelul funcțiilor cu un singur rând şi are sintaxa:

funcție (DISTINCT | ALL expresie1, expresie2, …)

Exemple :

LENGTH('BLAKE')
ROUND(1234.567*43)
SYSDATE

4) Forma IV este folosită pentru apelul funcțiilor de utilizator şi are sintaxa:

nume_schemă . nume_pachet. nume_funcție

Exemple:

aria_cercului(raza)
calcul_rate(nume_angajat)

5) Forma V este o combinație de mai multe expresii şi are sintaxa:
(*expresie*) | + | - | PRIOR *expresie* | *expresie1* * | / | - | || *expresie2*

Exemple:

('IONESCU' || ' PETRE')
LENGTH('BUCURESTI') * 57
SQRT(144) + 72
funcție_utilizator(TO_CHAR(sysdate,'DD-MMM-YY')

Expresiile pentru decodificarea unor valori folosesc sintaxa specială de tip **DECODE** de forma:

DECODE (expr, val1, rezultat1, val2, rezultat2, …. , valoare_asumată)

expr va fi evaluată şi apoi valoarea rezultată va fi comparată cu fiecarea dintre valorile *val1, val2, ….* . Dacă valoarea expresiei este egală cu una din valorile de comparație se va furniza valoarea rezultat (*rezultat1, rezultat2, ...*) care corespunde valorii de comparație. Dacă valoarea expresiei *expr1* nu este egală cu nici una din valorule de comparație se furnizează valoarea *valoare_asumată*. Dacă valoarea asumată este omisă atunci se furnizează valoarea **NULL**.

Exemplu:

```
DECODE (cod_funcție,10, 'programator',
                    20, 'cercetător',
                    30, 'vânzător',
                    40, 'operatorr',
                        'lipsă_funcție')
```

Exemplul de mai sus decodifică valoarea expresiei *cod_funcție*. Astfel dacă valoarea expresiei *cod_funcție* = 10 se furnizează funcția *programator*, dacă este = 20 se furnizează funcția *cercetător* şi aşa mai departe. *În* caz că expresia cod_funcție are a valoare care nu este egală cu nici una din valorile 10, 20, 30 sau 40 se furnizează ca rezultat valoarea *lipsă_funcție.*

B. O **listă de expresii** este o serie de expresii separate între ele prin virgulă şi închisă între paranteze rotunde şi poate conține pînă la maximum 1000 de expresii.

Exemplu:

(10, 20, 40)
('SCOTT', 'BLAKE', 'TAYLOR')
(LENGTH('MOOSE') * 57, -SQRT(144) + 72, 69)

3.5. CONDIȚIILE

Condiția este o combinație de una sau mai multe expresii şi operatori logici evaluată cu una din valorile TRUE, FALSE sau UNKNOWN. Condițiile se pot folosi în clauza WHERE a instrucțiunilor SQL **DELETE, SELECT şi UPDATE** sau într-una din clauzele **WHERE, START WITH, CONNECT BY** sau **HAVING a**le instrucțiunii **SELECT**.

Exemple:

1) Expresia *1=1* este evaluată ca TRUE

2) Expresia *NVL(sal, 0) + NVL(comm, 0) > 2500* adună valoarea sal cu comm şi evaluează rezultatul dacă este >= cu 2500000. Dacă sal sau comm sunt NULL se vor înlocui cu valoarea zero.

Condițiile au opt forme de prezentare.

Forma I este folosită pentru compararea unei expresii cu altă expresie sau cu o subcerere şi are sintaxa:

expresie1 = | != | <> | > | < | >= | <= expresi2 | (subcerere)
sau
listă_de_expresii = | != | <> (subcerere)

Forma II este folosită pentru compararea unei expresii sau a unei liste de expresii cu unul sau toți membrii unei liste de xepresii sau ai unei subcereri şi are sintaxa:

expresie1 = | != | <> | > | < | >= | <= ANY | SOME | ALL listă_de_expresii | (subcerere)
sau
listă_de_exp = | != | <> ANY | SOME | ALL (listă_expr1, listă_expr2, …) | (subcerere1, subcerere2, ….)

Forma III este folosită pentru căutarea unei expresii sau a unei liste de expresii dacă figurează într-o listă de expresii sau într-o subcerere şi are sintaxa:

expresie1 NOT IN listă_de_expresii | (subcerere)

sau

listă_de_exp NOT IN (listă_expr1, listă_expr2, …) | (subcerere1, subcerere2, ….)

Forma IV este folosită pentru testarea existenței sau inexistenței expresiei între două limite şi are sintaxa:

expresie1 NOT BETWEEN expr2 AND expr3

Forma V este folosită pentru testarea valorii NULL şi are sintaxa:

expresie1 IS NOT NULL

Forma VI este folosită pentru testarea existenței unui rând înt-o subcerere şi are sintaxa:

EXISTS (subcerere)

Forma VII este folosită pentru testarea egalității unei şir de caractere cu un format anume, cu un alt şir de caracter şi are sintaxa:

şir_ caracter1 NOT LIKE şir_cacrater2 ESCAPE 'caracter_de_schimbare'

Forma VIII este folosită pentru specificarea unei combinații de mai multe condiții şi poate avea sintaxele:

NOT condiție

sau

condiție1 NOT AND | OR condiție2

*3.6. DESCHIDEREA ŞI ÎNCHIDEREA UNEI SESIUNI DE LUCRU SQL*PLUS*

Pentru a avea acces la componentele unei baze de date (tabele, viziuni, clustere etc.), utilizatorul trebuie mai întîi să se conecteze la ea. La sfârşitul sau în timpul unei sesiuni SQL există posibilitatea deconectării de la baza de date. O primă conectare se face atunci cînd se începe lucrul cu SQL*Plus. Efectul comenzii sau utilizării meniului sistem îl constituie deschiderea unei sesiuni de lucru sub SQL*Plus.

Comanda are următoarele forme sintactice:

SQLPLUS [nume-utilizator[/parolă]
[@nume-bază-de-date]
[@nume-fişier]

[-SILENT]

SQLPLUS /NOLOG [-SILENT]

SQLPLUS -?

unde:
nume-utilizator şi parolă: sunt numele şi parola unui utilizator cu drept de acces la baza de date.
@nume-bază-de-date: este numele unei baze de date cu care se lucrează în rețea (este alt nume decât cel al bazei de date implicite, de pe calculatorul la care se lucrează).
@nume-fişier: reprezintă numele unui fişier de comenzi SQL care va fi fi rulat de SQL*Plus.
SILENT: are ca efect inhibarea facilității de afişare a tuturor informațiilor şi mesajelor furnizate de SQL*Plus
/NOLOG: lansează în execuție SQL*Plus dar nu face conectarea la o bază de date.
-?: are ca efect afişarea versiunii curente a componentei SQL*Plus, după care returnează controlul sistemului de operare

Dacă în linia de comandă se specifică *parola* şi *@nume-bază-de-date* atunci între ele nu trebuie să existe spațiu. *Parola* sau *@nume-bază-de-date* vor fi separate printr-un spațiu de *@nume-fişier*.

Conectarea utilizatorului la o bază de date

Dacă în timpul unei sesiuni de lucru SQL*PLUS se doreşte conectarea la o altă bază de date decât cea deschisă inițial se poate folosi comanda **CONNECT.**
Sintaxa acestei comenzi este:

CONN[ECT] [nume-utilizator[/parolă]]
[@nume-bază-de-date];

unde:
nume-utilizator şi parolă: sunt numele unui utilizator cu drepturi de acces la baza de date şi parola unui astfel de utilizator. Chiar dacă nu se specifică parola (în linia de comandă este opțională), ea este cerută de sistem printr-un mesaj explicit. Parola introdusă la mesajul sistemului va fi invizibilă.
@nume-bază-de-date: se specifică în cazul lucrului în rețea, cînd se doreşte conectarea la o altă bază de date decât cea aflată pe calculatorul la care se lucrează.

De remarcat faptul că în timpul unei sesiuni de lucru cu SQL*Plus se poate realiza conectarea la o bază de date, fără a mai fi necesară închiderea sesiunii.
Furnizarea parametrilor nume-utilizator şi parolă asigură protecția bazei de date împotriva accesului neautorizat.

Deconectarea utilizatorului de la baza de date

Deconectarea utilizatorului de la o bază de date se realizează prin comanda **DISCONNECT**.

Sintaxa acestei comenzi este:

DISC[ONNECT];

Comanda are ca efect deconectarea de la baza de date curentă, fără a închide sesiunea de lucru SQL*Plus.

Închiderea sesiunii de lucru SQL*Plus

Închiderea sesiunii de lucru SQL*PLUS şi predarea controlului sistemului de operare al calculatorului gazdă se realizează cu una din comenzile: **QUIT**, **EXIT** sau **^Z**.

Sintaxa acestor comenzi este:

QUIT;
EXIT:
^Z;

*3.7. ELEMENTE DE LUCRU CU SQL*PLUS*

A. Încărcarea şi executarea comenzilor

În exemplele ce se vor prezenta, referitor la utilizarea comenzilor SQL*Plus, se va folosi tabela Oracle **pers100** cu structura de mai jos:

```
CREATE TABLE pers100
	(CODPERS   NUMBER(5),
	NUME  VARCHAR2(30),
	PRENUME VARCHAR2(30),
	ZINAST NUMBER(2),
	LUNAST  NUMBER(2),
	ANAST NUMBER(4),
	STRADA VARCHAR2(40),
	NRSTR NUMBER(2),
	SECTOR  NUMBER(1),
	LOCNAST VARCHAR(20),
	FUNCTIA VARCHAR(15),
	SALARIU NUMBER(8),
	NUMAR_ACTIUNI  NUMBER (6))
```

PCTFREE 5 PCTUSED 75;

Comenzile se pot introduce pe *una* sau *mai multe linii.*

Exemplu:

Comanda

sql> select * from pers100;

este echivalentă cu:

sql> select
2 * from
3 pers100;

Introducerea comentariilor se poate realiza folosind:

- Comanda REMARK din SQL*Plus

Exemplu:

REMARK comentariu

Nu se vor introduce comentarii intre liniile aceleaşi comenzi SQL.

- Delimitatorii de comentariu din SQL /* … */

Exemplu:

/* comentariu */

- comentariile tip PL/SQL prefixate cu '- -'

Exemplu:

-- comentariu

Terminarea unei comenzi SQL se face cu: punct şi virgulă (;), slash (/) sau blank. Primele două forme cer SQL*Plus să execute imediat comanda, a treia formă nu. Comanda curentă poate fi rulată sau rerulată introducând comenzile RUN sau slash(/).

Se pot introduce şi blocuri PL/SQL care să fie executate. La sfârşitul blocurilor PL/SQL se vor insera două linii una conținând un *punct*, iar cealaltă un slash(/).

Exemplu:

```
declare
x number := 100;
begin
for i in 1 .. 10 loop
        insert into pers100  values
(10, 'ionel', 'marin', 10,7,1970, 'ion manolescu',2, 6, 'suceava');
end loop;
end;
/
```

Zona (aria) în care SQL*Plus memorează comenzile curente se numeşte *buffer-ul SQL*.

Pe lângă comenzile SQL şi PL/SQL se pot introduce şi comenzi SQL*Plus, care pot fi abreviate şi la o literă. Comenzile foarte lungi pot fi întrerupte şi continuate pe linia următoare. Întreruperea se marchează cu '-'. Oracle automat afişează semnul > (prompter) după care se introduce restul comenzii.

Exemplu:

sql> select * from -

>pers100;

Controlul listării rapoartelor lungi se poate face utilizând tasata Cancel sau setând variabila PAUSE pe ON.

Exemplu:

set pause 'text de atenționare'

set pause on

Această setare va determina ca sistemul să oprească afişarea la fiecare pagină şi să afişeze textul text de atenționare. Cu SET PAUSE OFF se revine la starea anterioară.

B. Editarea comenzilor SQL*Plus

Editarea în mod linie se realizează prin comenzile din tabelul 3.10.

Tabelul 3.10. Comenzile de editare a comenzilor SQL*Plus

Comanda	*Abreviația*	*Funcția*
APPEND text	*A text*	*Adaugă text la sfârşitul unei linii*
CHANGE /old/new/	*C /old/new/*	***Schimbă un text cu altul***
CHANGE /text	*C/text*	*Şterge textul unei linii*
CLEAR BUFFER	*CL BUFF*	*Curăță bufferul*
DEL	*Fără abreviație*	*Şterge o linie*
INPUT	*I*	*Adaugă una sau mai multe linii*
INPUT text	*I text*	*Adaugă o linie formată dintr-un text*
LIST	*L*	*Listează toate liniile din buffer*
LIST n	*L n sau n*	*Listează linia n*
*LIST **	*L **	*Listează linia curentă*
LIST LAST	*L LAST*	*Listează ultima linie*
LIST m n	*L m n*	*Listează liniile de la m la n*

Editarea comenzilor cu editorul sistemului se realizează cu comanda **EDIT.** Apare fereastra Editorului, în care se vor tasta instrucțiunile SQL dorite. Se salvează fişierul cu ***nume.SQL*** şi se execută cu comanda **@nume.**

C. Crearea, regăsirea, rularea şi modificaea fişierelor de comenzi.

Crearea fişierelor de comenzi se pot realiza prin:

- Salvarea conținutului bufferului cu comanda **SAVE.**

Exemplu:

SAVE nume_ fişier.SQL

Înainte de salvare se va lista bufferul cu comanda LIST pentru a verifica dacă instrucțiunile ce vor fi salvate sunt corecte.

- Utilizarea comenzii INPUT în corelație cu SAVE

Exemplu:

sql> clear buffer
sql> input
select * from pers100
sql> save comand1.sql
sql> @comand1

- Utilizarea editorului de sistem

SQL> EDIT nume_fişier

Regăsirea (citirea) fişierelor de comenzi se face cu comanda GET.

Exemplu:

SQL> GET edit3
SELECT * FROM PERS100;

Rularea fişierelor de comenzi se execută folosind comenzile:

START nume_fişier
@nume_fişier

Dacă avem mai multe fişiere de comenzi pe care vrem să le executăm secvențial, vom creea un fişier de comenzi conținând comenzile START corespunzătoare, după care pentru rulare vom activa acest ultim fişier.

Exemplu:

În fişierul FILE2 introducem comenzile:

START fiş1
START fiş2
START fiş3

Apoi cu una din comenzile START FILE2 sau @FILE2 vom activa aceste comenzi.

Modificarea fişierelor de comenzi se poate realiza folosind comanda **EDIT nume_fişier s**au comanda **GET** urmată de **SAVE nume_fişier REPLACE.**

D. Facilități pentru setarea fişierelor de comenzi

Următoarele facilități fac posibilă setarea unor fişiere de comenzi care să permită utilizatorilor să-şi introducă propriile valori pentru controlul

execuției comenzilor: definirea variabilelor de utilizator; ştergerea variabilelor de utilizator; substituirea valorilor în comenzi; folosirea comenzii START pentru furnizarea de valori în comenzi; crearea unor prompteri pentru introducerea valorilor.

1) Definirea variabilelor de utilizator se face explicit cu comanda **DEFINE** sau implicit prin utilizarea prefixării variabilelor cu două '**&**'.

Definirea, listarea şi ştergerea unei variabile utilizator în mod explicit se fac cu comenzile:

DEFINE nume_variabilă = "valoare variabila"

***Exemplu*:**

SQL > DEFINE variabila1 = "mihai"

2) Ştergerea variabilelor de utilizator se realizează prin utilizarea comenzii

UNDEFINE nume_varaiabilă.

***Exemplu*:**

SQL> UNDEFINE varaiabila1

3) Substituirea variabilelor este o tehnică prin care putem creea proceduri de lucru astfel încât să folosim acelaşi script (grup de instrucțiuni) pentru efectuarea unor funcții diferite pornind de la structura procedurii. Variabilele ce se substituie pot exista la momentul substituirii dacă anterior au fost create explicit cu comanda DEFINE sau implicit prin prefixare cu &. Există patru modalități de substituire a variabilelor: substituirea variabilelor *prefixate cu un '&'; s*ubstituirea variabilelor *prefixate cu două '&'; s*ubstituirea variabilelor de tip *&n* cu ajutorul comenzii *START; s*ubstituirea variabilelor folosind comenzile *PROMPT, ACCEPT şi PAUSE*

a) Substituirea variabilelor prefixate cu un '&'. Vom crea o procedură generalizată pentru calculul unor subgrupe statistice pe o coloană numerică.

***Exemplu**:*

Să se execute selecția din baza de date a valorilor salariilor aparținând aceleaşi funcții, iar din aceste grupe afişarea celor care sunt cele mai mici din grupă, ordonate descrescător.

select funcția , min(salariu) minimum
from pers100
group by funcția
order by min(salariu) desc;

În fraza SELECT de mai sus *funcția, min, salariu şi desc* pot fi definite ca variabile, ceea ce va permite ca să putem utiliza pentru grupare şi altă coloană, pentru calcul şi valorile max sau sum, iar pentru ordonare vom putea folosi şi valoarea ascendent. Pentru executarea acestei comenzi vom crea fişierul de comenzi *CALC&.SQL*, cu structura de mai jos:

/* --*/

/* &v1_col_grup = nume coloană din tabelă după valorile căreia se va face gruparea

/* &v2_tip_calc = tipul calculului: min, max, sum pentru un anumit grup de valori numerice

/* &v3_col_calc = numele coloanei de tip numeric după care se va face calculul

/*&v4_nume_col_calculat = numele coloanei, în listă, pe care se vor afişa valorile calculate

/ &v5_tip_sort = tipul ordonării(sortării), desc(descending) sau asc(ascending)*

```
SELECT &v1_col_grup,
       &v2_tip_calc(&v3_col_calc),
       &v4_nume_col_calculată
FROM   pers100
GROUP BY  &v1_col_grup
ORDER BY  &v2_tip_calc(&v3_col_calc,) &v5_tip_sort;
```

/* -- -- - - - - - - - - - - - --*/

După apelul fişierului cu comanda @calc& sistemul ne va cere succesiv să furnizăm valorile dorite pentru variabilele definite astfel:

Enter value for v1_col_grup: nume
Enter value for v2_tip_calc: min
Enter value for v3_col_calc: salariu
Enter value for v4_nume_col_calculat: MINIMUM
Enter value for v1_col_grup: nume
Enter value for v2_tip_calc: min
Enter value for v3_col_calc: salariu
Enter value for v5_tip_sort: desc

Iar rezultatul după executare comenzii va arăta ca mai jos:

```
NUME                 MINIMUM
-------------------------- ---------
ionescu              3500000
petrescu             2500000
mihai                1500000
```

b) Substituirea variabilelor prefixate cu două '&'. Pentru exemplificare vom crea, pornind de la fişierul de comenzi CALC&.SQL, fişierul de comenzi **CALC&&.SQL** în care variabilele vor fi prefixate cu doua caractere &, ca mai jos:

/* -- -- - - - - - - - - - - - -*/
/* &&v1_col_grup = nume coloană din tabelă după valorile căreia se va face gruparea
/* &&v2_tip_calc = tipul calculului: min, max, sum pentru un anumit grup de valori numerice
/* &&v3_col_calc = numele coloanei de tip numeric după care se va face calculul
/*&&v4_nume_col_calculat = numele coloanei, în listă, pe care se vor afişa valorile calculate
/* &&v5_tip_sort = tipul ordonării(sortării), desc(descending) sau asc(ascending) */

SELECT &&v1_col_grup,
&&v2_tip_calc(&&v3_col_calc),
&&v4_nume_col_calculat
FROM pers100
GROUP BY &&v1_col_grup
ORDER BY &&v2_tip_calc(&&v3_col_calc), &&v5_tip_sort

/* -- -- - - - - - - - - - - - - - -*/

La momentul executării fişierului de comenzi CALC&&.Sql sistemul va cere să introducem valorile dorite pentru variabilele definite la fel ca la apelul precedent cu deosebirea că valorile ce le vom furniza vor fi memorate de fiecare dată astfel că indiferent de câte ori apare o variabilă pentru ea se va furniza valoarea o singură dată. Orice execuție ulterioară a unui fişier de comenzi în care apare una din variabilele create anterior (definite implicit ca variabile utilizator) se vor folosi aceste valori. Execuția acestui fişier de comenzi va produce acelaşi rezultat ca şi execuția fişierului CALC&.SQL. Rulând comanda DEFINE vom găsi în sistem pe lângă alte variabile utilizator şi variabilele create prin execuția fişierului CALC&&.SQL.

Exemplu:

SQL> DEFINE
DEFINE _EDITOR = "Notepad" (CHAR)
DEFINE _RC = "1" (CHAR)
DEFINE V1_COL_GRUP = "nume" (CHAR)
DEFINE V2_TIP_CALC = "min" (CHAR)
DEFINE V3_COL_CALC = "salariu" (CHAR)
DEFINE V4_NUME_COL_CALCULAT = "minimum" (CHAR)
DEFINE V5_TIP_SORT = "desc" (CHAR)

Dacă vom dori rularea procedurii create anterior dar dând variabilelor alte valori, va trebui întâi să ştergem aceste variabile cu comanda UNDEFINE.

c) Substituirea variabilelor de tip &n cu ajutorul comenzii START. Pentru a nu mai furniza interactiv valori pentru variabilele utilizator la momentul execuției, le putem defini pe acestea sub forma ***&n*** , în care *n* ia valori începând cu **1,** *iar valorile lor vor fi furnizate ca parametrii de apel ai instrucțiunii START***.**

Astfel vom crea fişierul de comenzi **CALCSTART.SQL** de forma:

/ -*

/ &1 = nume coloană din tabelei după care se face gruparea*

/ &2 = tipul calculului: min, max, sum pentru un anumit grup*

/ &3 = numele coloanei de tip numeric după care se va face calculul*

/ &4 = numele coloanei pe care se vor afişa valorile calculate*

/ &5 = tipul ordonării(sortării), descending sau ascending*/*

select &1, &2(&3) &4

from pers100

group by &1

order by &2(&3) &5;

/ -*/*

Pentru execuție vom apela fişierul de comenzi CALCSTART.SQL cu comanda START în care vom furniza ca parametrii valorile dorite pentru cele 5 variabile de tip **&n** definite.

***Exemplu*:**

SQL> START CALCSTART nume min salariu MINIMUM desc

Această execuție va produce acelaşi rezultat ca şi execuțiile fişierelor de comenzi CALC&.SQL şi CALC&&.SQL.

Deosebirea este că nu se mai creează variabilele utilizator astfel că putem executa în mod generalizat procedura CALCSTART dând variabilelor orice alte valori logic acceptabile.

d) Crearea unor modalități interactive de comunicare cu calculatorul se realizează cu comenzile *PROMPT, ACCEPT şi PAUSE*, care sunt de fapt operații de intrare/ieşire standard. **PROMPT** permite trimiterea (scrierea) de mesaje la utilizator. **ACCEPT** permite preluarea (citirea) răspunsurilor de la utilizator. **PAUSE** permite introducerea unui moment de pauză pentru ca utilizatorul să citească mesajul şi să-şi formuleze răspunsul.

Aceste comenzi se folosesc în conjuncție cu INPUT şi SAVE, pentru a introduce valorile dorite cu comenzile de mai sus, şi respectiv pentru a le salva într-un fişier de comenzi, care să fie ulterior executat.

Exemplu:

SQL> Clear buffer

SQL> INPUT

PROMPT Introduceți un titlu format din maxim 30 caractere
PAUSE urmează introducerea titlului, apăsați RETURN
ACCEPT TITLUL_MEU PROMPT 'TITLUL: '
PAUSE urmează comanda de centrare a titlului, apăsați RETURN
TTITLE CENTER TITLUL_MEU SKIP 2
PAUSE urmează selectarea din baza de date, apăsați RETURN
SELECT codpers, nume , prenume from pers100;
SQL> SAVE cmdprompt
SQL> @cmdprompt

Rezultatul este:

```
Introduceți un titlu format din maxim 30 caractere
urmează introducerea titlului, apăsați RETURN
titlu:-----SELECTARE DIN BAZA DE DATE codpers, nume şi prenume ----
urmează comanda de centrare a titlului, apăsați RETURN
urmează selectarea din baza de date, apăsați RETURN
-----SELECTARE DIN BAZA DE DATE codpers, nume şi prenume ----
CODPERS   NUME                          PRENUME
--------- ----------------------------- ----------
 1        petrescu                      ion
 2        petrescu                      florea
 3        ionescu                       ion
 4        ionescu                       dumitru
 5        mihai                         florea
 6        mihai                         ion
6 rows selected.
```

e) Utilizarea comenzilor PROMPT şi ACCEPT în conjuncție cu substituirea variabilelor

În exemplele anterioare când s-au utilizat fişierele de comenzi CALC&.SQL şi CALC&&.SQL s-a văzut că sistemul pentru fiecare variabilă a creat câte un prompter de forma:

Enter value for nume_variabilă :

Ca atare se poate înlocui prompter-ul sistemului cu propriu prompter utilizând pentru aceasta comenzile PROMPT şi ACCEPT în fişierul de comenzi unde vrem să introducem o variabilă pentru care vom construi alt prompter decât cel de sistem.

Exemplu:

SQL> Clear buffer
SQL> INPUT
PROMPT Introduceți o valoare numerică pentru salariu
PROMPT De exemplu: 1500000, 2500000
ACCEPT var_salariu NUMBER PROMPT 'valoare salariu: '

SELECT codpers, nume, salariu from pers100
WHERE salariu = &var_salariu
SQL> save cmd2prompt
SQL> @cmd2prompt

Rezultatul este:

```
Introduceți o valoare numerică pentru salariu
De exemplu: 1500000, 2500000
valoare salariu: aaaa
“aaaa” is not a valid number
valoare salariu: 3500000
old    1: SELECT codpers, nume, salariu from pers100 WHERE salariu =
&var_salariu
new    1: SELECT codpers, nume, salariu from pers100 WHERE salariu =
3500000

CODPERS  NUME                          SALARIU
---------  ----------------------------- ---------
4          ionescu                      3500000
```

3.8. FORMATAREA REZULTATELOR

Limbajul SQL*PLUS permite proiectarea şi formatarea diverselor situaţii de ieşire. Aceste operaţii sunt posibile prin utilizarea unor comenzi pentru tratarea întreruperilor, comenzi pentru definirea titlurilor, definirea coloanelor, realizarea calculelor şi stabilirea diverselor opţiuni pentru poziţionarea unor arii de lucru pe ecran.

A. Tratarea întreruperilor

Întreruperea este un eveniment care se produce în timpul execuţiei unei comenzi SELECT, cum ar fi, de exemplu, apariţia sfîrşitului de pagină sau schimbarea valorii unei expresii.

Pentru a specifica evenimentele care determină o întrerupere şi acţiunea corespunzătoare SQL care se execută, se utilizează comanda BREAK. Ea poate specifica mai multe evenimente care generează o întrerupere. Evenimentele sunt reţinute într-o ordine numită "ierarhie de întrerupere". La un moment dat,"se poate executa doar o singură comandă **BREAK**.

Comanda **BREAK** are următoarele forme sintactice:

BRE[AK] ON {expr | ROW | PAG[E] | REPORT}
[SKI[P] n | [SKIP]PAGE]
[NODUP[LICATES] | DUP[LICATES]];
BRE[AK];

unde:
ON expr determină o întrerupere cînd se schimbă valoarea expresiei expr; expr este fie o expresie care implică una sau mai multe coloane dintr-o tabelă, fie o etichetă ataşată unei coloane declarată in comanda **SELECT** sau **COLUMN**.
Dacă **ON** expr apare de mai multe ori în comandă, atunci expresiile respectă "ierarhia de întrerupere", aceasta fiind chiar ordinea în care sunt specificate expresiile. Cînd se foloseşte **ON** expr, trebuie utilizată şi clauza **ORDER BY**, pentru a ordona rândurile din "ierarhia de întrerupere*. Ordinea de apariţie a expresiilor expr în comanda **BREAK ON** trebuie să rte aceeaşi cu cea prezentă în **SELECT...GRDER BY**:
ON ROW determină o întrerupere cînd se selectează un rând cu **SELECT**. Această întrerupere este reţinută la sfîrşitul ierarhiei de
întrerupere;
ON PAGE determină o întrerupere la sfârşitul fiecărei pagini;
ON REPORT determină o întrerupere la sfârşitul raportului sau cererii, întrerupere care este reţinută la începutul ierarhiei de întrerupere.
SKIP determină saltul peste n linii, înainte de a se tipări linia
asociată întreruperii respective.
PAGE sau **SKIP PAGE** determină saltul la o pagină nouă înainte de a tipări linia asociată respectivei întreruperi.
NODUPLICATES determină tipărirea de spaţii cînd valorile de JJS coloana de întrerupere sunt identice. **DUPLICATES** determină tipărirea valoarii coloanei de întrerupere în fiecare rând selectat. Valoarea **NODUPLICATES** este implicită.
Comanda **BREAK** fără clauze indică poziţia întreruperii curente.
Exemple:
1) Să fie definită o întrerupere generată de schimbarea valorilor coloanei FUNCT. La apariţia acestui eveniment să fie afişate două linii vide.

SQL> BREAK ON FUNCT SKIP 2;
SQL> SELECT MARCA,NUME,CODD,
2 SALA, VENS
3 FROM SALARIAŢI
4 ORDER BY FUNCT;

MARCA	NUME	FUNCT	CODD	SALA	VENS
7000	ION ION	DIRECTOR	100000	45000	40000
2550	FRINCU ION	SEF DEP	160000	36000	37000
1000	COMAN RADU		130000	35000	2500
2500	VLAD VASILE		160000	36500	1500
4000	PAUL ŞTEFAN		160000	35000	5600
3755	DORU DAN		130000	36500	5500

1111	AVRAM ION VINZATOR	100000	21200	1000
1680	RADU ION	130000	20750	3000
3700	MÂNU DAN	160000	27500	2500
2553	AILENEI FLORIN	120000	25000	400
3760	SANDU ION	130000	25600	0
3770	CARMEN ANA	130000	26500	
2554	DARIAN GEO	120000	26000	2000
3759	ALEXE IOAN	160000	25700	
3500	DAN ION	160000	24500	350
2650	VLAD ION	120000	25060	3500
1222	BARBU DAN	120000	20750	2000

2) Să fie definită o întrerupere la apariția unei schimbări a valorii coloanei ODS din tabela SALARIAȚI. În momentul realizării evenimentului să se realizeze salt la pagină nouă.

SQL> SET PAGESIZE 11
SQL> BREAK ON CODS SKIP PAGE;
SQL> SELECT * FROM SALARIAT
2 ORDER BY CODS;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
1111	AVRAM ION	VINZATOR	100000	21200	1000	1000
2650	VLAP ION	VINZATOR	120000	25060	3500	
1222	BARBU DAN	VINZATOR	120000	20750	2000	

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
2550	FRINCU ION	SEF DEP	160000	36000	37000	2500
3500	DAN ION	VINZATOR	160000	24500	350	
1680	RADU ION	VINZATOR	130000	20750	3000	

MARCA	NUME FUNCT	CODD SALA	VENS	CODS	VENS	CODS
2553	AILENEI FLOR	VINZATOR	120000	120000	250000	2000
2554	DARIAN GEO	VINZATOR	260000	4000		

MARCA	NUME	FUNCT CODD	SALA	VENS	CODS	
7000	ION ION	DIRECTOR	100000	45000	40000	8000

9records selected.

B. Tipărirea titlurilor

Pentru afişarea unui titlu la sfîrşitul sau la începutul fiecărei pagini se utilizează comanda BTITLE, respectiv TTITLE. Comanda BTITLE are următoarele sintaxe:

BTI[TLE] [COL[UMN] n1 [SKIP [n]] [TAB n] [LEFT | RIGHT | CENTER] [FORMAT char] [char | var]...;

BTI[TLE] {OFF | ON};
BTI[TLE] text;
BTI[TLE]

unde:
COL[UMN] n determină saltul la coloana n a liniei curente.
Prin SKIP n se determină saltul Ia începutul liniei curente, de n ori. Dacă n este omis, se sare o singură dată, iar dacă n este zero, se realizează întorcerea ia începutul liniei curente.
TAB n are a efect saltul a n coloane (prin coloană înțelegîndu-se nu o coloană de tabelă, ci poziția cursorului) înainte dacă n este pozitiv sau înapoi dacă n este negativ.
Clauzele LEFT, RIGHT, CENTER determină alinierea la stânga, la dreapta respectiv afişarea centrata a datelor din linia curentă. Următoarele date sunt aliniate ca an grup, de la începutul până la sfârşitul comenzii PRINȚ sau la următorul LEFT, CENTER, RIGHT sau COLUMN. CENTER şi RIGHT folosesc valoarea returnată de comanda SET LINESIZE pentru a calcula poziția următorului articol.
FORMAT char specifică un model de format pentru articolul de date care urmează; acest format se menține până la întîlnirea unei alte clauze FORMAT sau până la sfârşitul comenzii. De reținut că doar un singur model de format poate fi activ la un moment dat. Dacă nici un format potrivit nu are efect, atunci valorile sunt tipărite în conformitate cu formatul specificat în SET NUMFORMAT iar dacă SET UNFORMAT nu a fost utilizat, valorile vor fi tipărite cu formatul implicit.

Pentru specificarea titlurilor pot fi folosite constante (char) şi variabile (var), ele fiind poziționate şi formatate aşa cum se specifică în clauzele comenzii.

Existența unui separator indică începutul unor linii noi, iar doi separatori pe acelaşi rând introduc o linie vidă. Inițial, caracterul de separare este."!", însă el poate fi schimbat cu SET HEADSEP.

SQL*PLUS interpretează comanda BTITLE îh forma nouă dacă primul cuvânt după numeie comenzii este numele unei clauze valide (LEFT, SKIP, COL etc.).

Clauzele ON şi OFF au ca efect apariția (ON) sau nu (OFF) a titlului.

BTITLE text afişează centrat textul specificat.
Comanda BTITLE fără clauze specifică titlul curent.
Exemple:

1) Să se afişeze la sfârşitul unui raport final cu privire la produsele din depozitul cu codul 100000, în partea stângă şirul de caractere "Data:" iar în partea dreaptă "Semnătura:".

SQL> SET PAGESIZE 11
SQL> BTITLE LEFT Data: RIGHT Semnătura:;
SQL> SELECT • FROM PRODUSE
2 WHERE CODD=100000;

CODD	CODP	DENP	STOC	DATA CRT	UM
100000	D4	SCAUN	36	10-SEP-05	BUC
100000	A3	FOTOLIU	27	15-SEP-05	BUC
100000	A7	MASA	23	05-SEP-05	BUC

Data: Semnatura:

2) Să se afişeze la sfârşitul unui raport privind situația produselor din depozitul 100000, începând din coloana 11, şirul de caractere "OBSERVATIT.

SQL> BTITLE COL 11 OBSERVAȚII;
SQL> SELECT * FROM PRODUSE
2 WHERE CODD-100000;

CODD	CODP	DENP	STOC	DATACRT	UM
100000	166666	PLACAJ 2/2	100	12-JUL-05	MP
100000	144444	SCAUN D4	36	12-JUL-05	BUC
100000	111111	MESE 15/20	7	27-JUN-05	BUC
100000	122222	FOTOLIU A3	12	01-JUL-05	BUC
100000	133333	CANAPEA A7	6	18-JUL-05	BUC
100000	155555	BIROU C6X4	9	29-JUL-05	BUC

OBSERVAȚII

3) Să se afişeze centrat, la sfîrşitul unui raport privind produsele din depozitul cu codul 100000, şirul de caractere"Depozitul_MOBILA/100000".

SQL> BTITLE CENTER Depozitul_MOBILA/100000
SQL> SELECT CODD "Cod depozit",
2 DENP "Denumire",
3 CODP "Cod produs",
4 STOC "Stoc",DATACRT "Data",
5 UM
6 FROM PRODUSE
7 WHERE CODD=100000;

Cod dep	Denumire	Cod produs	Stoc	Data	UM

100000	PLACAJ 2/2	166666	100	12-JUL-05	MP
100000	SCAUN D4	144444	36	12-JUL-05	BUC
100000	MESE 15/20	111111	7	27-JUN-05	BUC
100000	FOTOLIU A3	122222	12	01-JUL-05	BUC
100000	CANAPEA A7	133333	6	18-JUL-05	BUC
100000	BIROU C6X4	155555	9	29-JUL-05	BUC

Depozitul MOBILA/100000

4) Să se afişeze specificaţiile curente pentru BTITLE. Ultima comandă primită în sistem se prezintă astfel:

SQL> BTITLE COL 40 Total RIGHT Semnătura;
SQL> BTITLE;

btitle ON and is the following 28 characters: COL 40 Total RIGHT Semnătura

Comanda **TTITLE** determină afişarea unui titlu la începutul paginii şi are următoarele sintaxe:

TTI[TLE] [COL[UMN] n] [SKIP [1 | n] [TAB n]
[LEFT | RIGHT | CENTER]
[FORMAT char] [char | var]
TTI[TLEI text;
TTI[TLE];

Clauzele comenzii TTITLE au aceeaşi semnificaţie ca la BTITLE.
Comanda TTITLE fără clauze determină afişarea titlului curent.
Exemplu:
Să se afişeze la începutul unui raport privind comenzile cu date mai mică de 30 septembrie 2005, următoarele şiruri de caractere: "SITUAŢIA COMENZILOR" şi "LA DATA DE 30-SEP-05". Cele două şiruri se vor afişa centrat, pe două linii.

SQL> SET PAGESIZE 11 SQL> SET LIN 45
SQL> COLUMN NRCOM FORMAT 99999999
SQL> COLUMN NRCOM JUSTIFY CENTER
SQL> COLUMN CODP FORMAT 99999999
SQL> COLUMN CODP JUSTIFY CENTER
SQL> COLUMN VALOARE FORMAT 99999999999
SQL> COLUMN VALOARE JUSTIFY CENTER
SQL> SET SPACE 7
SQL> TTITLE 'SITUAŢIA COMENZILOR LA
DATA DE 30-SEP-05"
SQL> SELECT
2 NRCOM,CODP,CANT*PRET VALOARE

3 FROM COMENZI
4 WHERE DATAL<='30- SEP-05;

SITUATIA COMENZILOR LA DATA DE 30- SEP-05

NRCOM	CODP	VALOARE
211111	233333	244444
255566	133333	144444
166666	222222	320000
27000	375000	282000

Anularea opțiunilor

în vederea anulării opțiunilor se utilizează comanda:

CL[EAR] option

Este anulată opțiunea specificată prin option, după cum urmează:

BRE[AKS] anulează întreruperea indicată prin comanda BREAK;
BUFF[ER] determină ştergerea textului din buffer-ul curent;
COL[UMNS] anulează opțiunile indicate în comanda COLUMN;
COMP[UTES] anulează opțiunile indicate prin comanda COMPUTE;
SCR[EEN] determină ştergerea ecranului, iar SQL determină ştergerea buffer-ului SQL;
TIMI[NG] şterge toate zonele create cu comanda TIMING.

Exemple:

1) Să se anuleze întreruperile.

SQL> CLEAR BREAKS;

2) Să se anuleze definițiile de coloană.

SQL> CLEAR COLUMNS;

Descrierea coloanelor

Pentru specificarea modului de formare a unei coloane şi a capului de coloană într-o situație, se utilizează comanda COLUMN. Sintaxa ei este:

COL[UMN] (col | expr) [ALI[AS] sinonim] CLE[AR] | DEF[AULT]] [COLOR {culoare| variabila-culoare}] [FORMAT] format] [HEA[DING] text
[JUS[TIFY] {L[EFT | C[ENTER] | R[IGHT]] [LIKE {expr | etichetă}] [LINEAPP {LINE | MARK | BOTH}]
[NEW_VALUE variabila] [NU[LL] char] [NOPRI[NT] | PRINT]] [OLD_VALUE variabila]
[ON | OFF]
[PATTERN {număr-de-model | variabila-model}
[WRA[PPED]! WOR[D,... WRAPPED] |
TRU[NCATED]]
...;

unde:
col sau expr au rolul de a identifica coloana sau expresia la care se referă comanda. Comanda COLUMN trebuie să se refere la o coloană sau la o expresie utilizat-a în comanda SELECT. Dacă comanda COLUMN se referă la o expresie, aceasta trebuie să fie specificată în acelaşi mod în care a fost specificată în SELECT. De exemplu, dacă în SELECT expresia este 'A.+B', nu se poate folosi, în COLUMN, T+A'. Dacă sunt selectate din tabele diferite coloane cu acelaşi nume, atunci comanda COLUMN se va aplica tuturor tabelelor ce conțin acea coloană.
Clauza ALIAS pune în corespondență numele coloanelor cu sinonimul specificat.
Clauza CLEAR are ca efect ştergerea în totalitate a definiției coloanei.
Clauza FORMAT specifică formatul de apariție ai coloanei. Lățimea implicită a coloanei este chiar lățimea coloanei definite în baza de date. Ea poate fi schimbată cu valoarea n, folosind "FORMAT An". Lățimea unei coloane numerice este implicit valoarea data de NUMWIDTH, ea putând fi schimbată cu ajutorul clauzei FORMAT.
Clauza HEADING defineşte capul coloanei, care implicit este col sau expr. Dacă textul din definirea coloanei conține spații sau semne de punctuație, acestea trebuie puse între ghilimele. Fiecare semn "I" din text are ca efect începerea unei linii noi.
Clauza JUSTIFY aliniază capul coloanei. Implicit, alinierea se face la dreapta pentru o coloană numerică şi la stînga pentru alte tipuri de coloane.
Clauza LIKE determină copierea specificațiilor altei coloane sau expresii (specificații deja definite prin altă comandă COLUMN).
Prin folosirea clauzei NEWLINE se trece la o linie nouă, înainte de a afişa valorile coloanei.
Clauzele PRINT şi NOPRINT au ca efect tipărirea sau nu a coloanei.
Dacă se utilizează clauza NULL, textul va fi afişat chiar dacă conține o valoare nulă. Valoarea implicită este un şir de blancuri.
Clauzele OFF sau ON determină folosirea, respectiv nefolosirea formatului specificației de ieşire pentru o coloană, fără a-i afecta conținutul.
Clauza WRAPPED are ca efect scrierea pe linia următoare a caracterelor care nu au încăput pe prima linie.
Clauza WORD_WRAPPED are efect asemănător cu cel al clauzei WRAPPED, cu deosebirea că determină scrierea întregului cuvânt pe linia următoare şi nu pe două rânduri ca la WRAPPED.
Clauza TRUNC determină trunchierea valorii.
Clauza COLOR specifică culoarea utilizată pentru afişarea valorilor coloanei, într-o reprezentare liniară sau prin benzi. Variabila CLR n (n=l,60)

se referă la valoarea curentă a culorii. Setarea se face prin comanda SET CLR n.
Clauza LINEAPP stabileşte modul de subliniere a coloanei. Utilizând LINE dreapta apare continuă. Folosind MARK se schițează doar puncte, iar BOTH realizează ambele modalități: linie şi puncte.
Clauza PATTERN specifică modelul pe care-1 va avea dreptunghiul/banda într-un grafic de tip benzi sau sectorul într-un grafic de tip cerc. număr-de-model este cuprins între 1 şi 16 şi reprezintă un model. Variabila PAT n, unde ns[l,30] se referă la poziția curentă a modelului. Semnificația valorilor din număr-de-model şi a valorilor PAT n sunt precizate în documentație.

Numărul de comenzi COLUMN folosite indică faptul că se lucrează cu diverse coloane sau expresii utilizate în comanda SELECT. Pentru o coloană sau expresie pot exista mai multe comenzi COLUMN. În cazul în care mai multe comenzi COLUMN folosesc aceeaşi clauză pentru aceeaşi coloană, se va lua în considerare doar ultima comandă.

Comanda COLUMN fără clauze are ca efect tipărirea definițiilor coloanei curente. Dacă această comandă are doar clauzele col şi expr, efectul ei constă în indicarea definirii coloanei existente.

Trebuie precizat că în prima comanda COLUMN, expresia trebuie să aibă aceeaşi formă ca şi în SELECT. În caz contrar, SQL*Plus nu va putea executa comanda COLUMN pentru coloana respectivă.

Exemple:

1) Să se definească coloana NUME pe o lățime de 25 de caractere iar capul de coloană să fie scris pe două rânduri, astfel: NUME PRODUS

SQL> COLUMN NUME FORMAT A25 HEADING 'NUME PRODUS';
5QL> SELECT DISTINCT(DENP) NUME
2 FROM PRODUSE;

NUME PRODUS
MESE 15/20
FOTOLIU
2 records selected.

2) Să se afişeze sinonimul SALARIU pentru coloana definită de expresie aritmetică SALA+VENS. Coloana va fi scrisă cu formatul $99,999.99.

SQL> COLUMN SALA+VENS ALIAS SALARIU
SQL> COLUMN SALARIU FORMAT $99,999.99
SQL> SELECT SALA+VENS SALARIU,
2 MARCA, NUME
3 FROM SALARIAȚI;

SALARIU	MARCA	NUME
$22,200.00	1111	AVRAM ION
$22,750.00	1222	BARBU DAN

2 records selected.

3) Să se definească coloana DENP de 16 caractere alfanumerice, pentru care se specifică sinonimul DENUMIRE . In capul de coloană se va afişa textul "Denumire produs".

SQL> COLUMN DENP FORMAT A16
SQL> COLUMN DENP ALIAS DENUMIRE
SQL> COLUMN DENUMIRE HEADING "Denumire produs"
SQL> SELECT * FROM PRODUSE;

CODD	CODP	Denumire produs	STOC	DATACRT	UM
100000	166666	PLACAJ 2/2	100	12-JUL-92	MP
100000	122222	FOTOLIU A3	7	27-JUN-92	BUC
100000	133333	CANAPEA A7	36	12-JUL-92	BUC
100000	155555	BIROU C6X4	12	01-JUL-92	BUC
100000	144444	SCAUN D4	6	18-JUL-92	BUC
100000	111111	MESE 15/20	9	29-JUL-92	BUC

6 records selected.

4) Să se definească coloana DENUM2 cu un format alfanumeric
de trei caractere. Specificațiile pentru această coloană sunt
copiate pentru coloana DENP.

SQL> COLUMN DENUM2 FORMAT A3
SQL> COLUMN DENP LME DENUM2
SQL> SELECT * FROM PRODUSE 2 WHERE CODP<138333;

CODD	CODP	DEN	STOC	DATACRT	UM
100000	111111	MES	7	27-JUN-05	BUC

5) Să se definească coloana ADRESA pe 21 de caractere, utilizând clauza WRAPPED.

SQL> COLUMN FORMAT A21 ADRESA WRAPPED
SQL> SELECT STR11 '-' 11NR11 '-' 11 LOC ADRESA 2 FROM CLIENȚI;

ADRESA
MOŞILOR- 104-BUCURESTI
DOROBANȚI- 18-BUCURESTI
GOLENTINA-221-BUCURESTI
EMINESCU-44-BUCURESTI
4 records selected.

D.Realizarea de calcule

SQL*Plus permite realizarea unor calcule cu rândurile selectate din tabele. Pentru aceasta se utilizează comanda COMPUTE, care are următoarea sintaxă:

COMP[UTE] [AVG | COU[NT] | MAX[IMUM] |
MIN[IMUM] NUMBER) ISDT | SUM | VAR[IANCE}]
OF { expresiei etichetă),...
ON [expresiei etichetă IPAGE1 REPORT! ROW];

Semnificația clauzelor este:

Clauza	Semnificația	Tipul coloanei Tipuri de date
AVG	Valoare medie	Numeric
COUNT	Contorizare valori nule	Toate tipurile
MAXIMUM	Valoare maximă	Numeric şi caracter
MINIMUM	Valoare minimă	Numeric şi caracter
NUMBER	Contorizare rânduri	Toate tipurile
STD	Abatere standard	Numeric
SUM	Suma valorilor nenule	Numeric
VARIANCE	Dispersia	Numeric

Dacă se specifică mai multe funcţii, nu trebuie separate între ele prin virgulă.

OF precizează coloana sau expresia ale căror valori se vor supune calculului. Acestea trebuie să apară în comanda SELECT, altfel comanda COMPUTE le va ignora. Dacă se doreşte ca valoarea calculată să nu fie afişată pe ecran se va utiliza clauza **NON PRINT** ataşată coloanei respective cu comanda **SELECT.**

ON specifică un element care va fi utilizat ca generator de întrerupere. El poate fi: expresie, etichetă, pagină (PAGE), raport (REPORT), linie (ROW). De câte ori se va schimba valoarea acestuia, se va declanşa automat recalcularea funcției definite de **COMPUTE.**

În cazul în care se dă comanda fără nici o funcție, se vor afişa specificațiile de calcul definite.

Secvența de instrucțiuni care se va utiliza, în general, va fi:

SQL* BREAK ON expresiei
SQL> COMPUTE clauza OF expresie2 ON expresiei;
SQL> SELECT

***Exemple*:**

1) Să se editeze situația finală cu următorul format:

SITUAȚIE SUM(SALA) FUNCȚIE

SQL> SET PAGESIZE 22
SQL>RUN
1 SELECT SUM(SALA),FUNCT FROM SALARIAȚI

2* GROUP BY FUNCT

SITUAȚIE

SUM(SALA)	FUNCT
45000	DIRECTOR
179000	SEF DEP
268560	VINZATOR

2) Să se editeze o situație finală cu salariile grupate pe funcții şi depozite.

SQL> RUN
1 SELECT FUNCT,
2 SUM(DECODE(CODD,100000,SALA,0)) "DEP 10",
3 SUM(DECODE(CODD,130000,SALA,0)) "DEP 13",
4 SUM(DECODE(CODD,160000,SALA,0)) "DEP 16"
5 FROM SALARIAȚI
6 GROUP BY FUNCT;

SITUATIE

FUNCT	DEP	DEP	DEP
DIRECTOR	45000	0	0
SEF DEP	0	71500	107500
VINZATOR	21200	72850	77700

Data: Semnătura:

3) Să se afişeze o situație finală cu salariile grupate pe funcții şi depozite. De asemenea, să se introducă o coloană cu suma salariilor pe fiecare funcție.

SQL>RUN
1 SELECT FUNCT,
2 SUM(DECODE(CODD,100000,SALA,0)) "DEP 10",
3 SUM(DECODE(CODD,130000,SALA,0)) "DEP 13",
4 SUM(DECODE(CODD,160000,SALA,0)) "DEP 16",
5 SUM(SALA)
6 FROM SALARIAȚI
7 GROUP BY FUNCT;

SITUATIE

FUNCT	DEP 10	DEP 13	DEP 16	SUM(SALA)
DIRECTOR	45000	0	0	45000
SEF DEP	0	71500	107500	179000
VINZATOR	21200	72850	77700	268500

Data: Semnătura:

4) Să se afişeze o situaţie finală cu salariile grupate pe funcţii şi depozite. De asemenea, să se facă totalul salariilor pe fiecare depozit şi fiecare funcţie.

SQL> BREAK ON DUMMY;
SQL> COMPUTE SUM OF "DEP 100000" ON DUMMY;
SQI> COMPUTE SUM OF "DEP 130000" ON DUMMY;
SQL> COMPUTE SUM OF "DEP 160000* ON DUMMY;
SQL> COMPUTE SUM OF "TOTAL" ON DUMMY;
SQI> COLUMN DUMMY NOPRINT;
SQL> TTITLE CENTER "SITUAȚIA SALARATILOR"
SKIP CENTER "PE DEPOZITE /FUNCȚII"
SKIP CENTER " "
SQL> RUN
1 SELECT FUNCT,
2 SUM(DECODE(CODB,lG0000,SALA,0)) "DEP 10",
3 SUM(DECODE(CODD,130000,SALA,0)) "DEP 13",
4 SUM(DECODE(CODD,160000,SALA,0)) "DEP 16",
5 SUM(SALA,)
6 SUM(0) DUMMY
7 FROM SALARIAȚI
8 GROUP BY FUNCT

SITUATIA SALARIATILOR
PE DEPOZITE / FUNCTII

FUNCT	DEP 10	DEP 13	DEP 16	SUM(SALA)
DIRECTOR	45000	0	0	45000
SEF DEP	0	71500	107500	179000
VINZATOR	21200	72850	77700	268500
	66200	144350	185200	

Data: Semnătura:

CAPITOLUL 4. CREAREA UNEI BAZE DE DATE PRIN COMENZI SQL

4.1. TIPURI DE UTILIZATORI AI BAZELOR DE DATE ORACLE

În funcție de volumul activităților implicate de administrarea unei baze de date Oracle, sarcinile de administrare pot fi repartizate pe mai multe categorii de utilizatori ai bazei de date.

Administratorul bazei de date (DBA) este, în funcție de complexitatea şi mărimea unei baze de date, o persoană sau mai multe persoane, care să execute următoarele sarcini administrative:

- Instalarea şi dezvoltarea sever-ului Oracle;
- Alocarea memoriei sistemului şi planificarea cerințelor viitoare de memorie ale acestuia;
- Crearea bazei de date şi a obiectelor acesteia (tabele, viziuni, indecşi);
- Modificarea structurii bazei de date în funcție de cerințele dezvoltatorilor de aplicații;
- Definirea utilizatorilor bazei de date şi întreținerea sistemului de securitate;
- Controlul şi monitorizarea accesului utilizatorilor la baza de date;
- Monitorizarea şi optimizarea performanțelor bazei de date;
- Definirea şi asigurarea politicii de salvarea sau copiere (backup) şi refacere (recovery) a bazei de date;
- Arhivarea datelor;
- Asigurarea legăturii cu firma Oracle pentru suportul tehnic şi licența de utilizare a produselor Oracle.

Dezvoltatorii de aplicații proiectează şi implementează aplicații cu baze de date Oracle, executând următoarele sarcini:

- Proiectarea şi dezvoltarea unei aplicații, precum şi a structurilor de date ale acesteia;
- Estimarea cerințelor de memorie pentru aplicație;
- Definirea modificărilor structurilor de date ale unei aplicații;
- Transmiterea tuturor informațiilor despre activitățile de mai sus către administratorul bazei de date;
- Stabilirea măsurilor de securitate pentru aplicație.

Administratorul de aplicații se ocupă cu administrarea unei aplicații;

Utilizatorii finali ai bazei de date au acces la baza de date prin intermediul unei aplicații sau a instrumentelor Oracle, executând în special următoarele activități:

- Adăugarea, modificarea şi ştergerea datelor din baza de date în concordanță cu drepturile de acces pe care le are;
- Generarea unor rapoarte cu datele din baza de date.

Administratorul de rețea este responsabil cu administrarea produselor Oracle de rețea.

Pentru a putea executa sarcinile de administrare a unei baze de date o persoană trebuie să aibă privilegii (drepturi) atât la nivelul bazei de date Oracle, cât şi la nivelul sistemului de operare al serverului pe care se află baza de date. Prin urmare un administrator trebuie să aibă:

- *Cont de administrator pentru sistemul de operare*, care să-i permită să execute comenzile sistemului de operare;
- *Cont de administrator Oracle* definit de două conturi de utilizator (SYS şi SYSTEM cu parolele CHANGE_OF_INSTALL şi respectiv MANAGER);
- *Rol de DBA*, care este creat automat la momentul creării unei baze de date Oracle. Acest rol conține toate privilegiile bazei de date Oracle.

Datorită faptului că un administrator execută activități pe care un utilizator obişnuit nu le poate executa este necesar ca acesta să poată fi autentificat înainte de a executa activitățile de administrare. Pentru autentificarea administratorului există două metode: autentificarea folosind sistemul de operare şi autentificarea folosind fişierul de parole.

Autentificarea folosind *sistemul de operare* se face utilizând două conturi de administrator: *OSOPER (*sub care poate executa STARTUP, SHUTDOWN, ALTER DATABASE OPEN/MOUNT, ALTER DATABASE BACKUP, ARCHIVELOG şi RECOVER) şi contul *OSDBA (*care conține toate privilegiile de sistem cu opțiunea ADMIN OPTION, precum şi rolul OSOPER). Sub contul OSDBA se poate executa comanda CREATE DATABASE.

Autentificarea folosind *fişierul de parole* permite definirea parolelor de acces pentru fiecare utilizator. După stabilirea unui utilizator ca administrator, de exemplu utilizatorul SCOTT cu parola TIGER, acestuia îi va fi atribuit unul din privilegiile SYSDBA sau SYSOPER, cu comanda GRANT. După aceasta utilizatorul SCOTT se va conecta la baza de date ca SYSDBA sau SYSOPER cu comanda CONNECT.

***Exemplu*:**

GRANT SYSDBA TO scott

GRANT SYSOPER TO scott
CONNECT scott/tiger AS SYSDBA
CONNECT scott/tiger AS SYSOPER

Administrarea fişierului cu parole include operaţiile de *definire* a fişierului cu parole, *setarea* parametrului de iniţializare a bazei de date REMOTE_LOGIN_PASSWORDFILE, *adăugarea de utilizatori* în acest fişier şi *întreţinerea* fişierului cu parole.

Crearea fişierului cu parole se execută cu utilitarul ORAPWD, care are trei parametrii: FILE, PASSWORD şi ENTRIES, dintre care primii doi sunt obligatorii, iar ultimul este opţional. Aceşti parametrii definesc *numele fişierului cu parole*, *parola* pentru utilizatorul SYS şi respectiv *numărul de utilizatori* care pot executa activităţi de administrator (DBA). *Setarea parametrului de iniţializare* REMOTE_LOGIN_PASSWORDFILE cu una din valorile NONE, EXCLUSIVE şi SHARED permite utilizarea fişierului cu parole.Valoarea NONE determină ca baza de date Oracle să funcţioneze fără fişier de parole, valoarea EXCLUSIVE determină ca fişierul de parole să fie folosit exclusiv de către o singură bază de date, iar valoare SHARED determină ca fişierul de parole să fie folosit de către mai multe baze de date.

Pentru a avea un grad mare de securitate pentru baza de date, va trebui ca imediat după crearea fişierului cu parole parametrul de iniţializare REMOTE_LOGIN_PASSWORDFILE să fie setat pe valoarea EXCLUSIVE.

Adăugarea de utilizatori în fişierul cu parole se face la momentul atribuirii privilegiilor SYSDBA sau SYSOPER unui anumit utilizator.

***Exemplu*:**

1. *Se creează fişierul cu parole conform indicaţiilor de mai sus;*
2. *Se setează parametrul de iniţializare REMOTE_LOGIN_PASSWORDFILE cu valoarea EXCLUSIVE;*
3. *Se conectează utilizatorul SYS, cu parola CHANGE_OF_INSTALL ca SYSDBA folosind comanda*
   CONNECT SYS/change_of_install AS SYSDBA
4. *Se porneşte o instanţă şi se creează o bază de date, dacă este necesar, sau se montează şi se deschide o bază de date existentă;*
5. *Se creează utilizatorii care se doresc a fi administratori şi care să fie adăugaţi în fişierul cu parole, folosind comanda*
   CREATE USER user1 IDENTIFIED BY parola1
6. *Se atribuie unul din privilegiile SYSDBA sau SYSOPER acestui utilizator cu una din comenzile:*
   GRANT SYSDBA TO user1 sau GRANT SYSOPER TO user1

7. *Utilizatorul USER1 este adăugat în fişierul cu parole şi se poate conecta acum ca SYSDBA sau SYSOPER cu acest nume de utilizator în loc de numele SYS, folosind una din comenzile:*

CONNECT USER1/parola1 AS SYSDBA *sau*

CONNECT USER1/parola1 AS SYSOPER

Listarea membrilor fişierului cu parole se face din viziunea $PWFILE_USERS folosind comanda

SELECT * FROM V$PWFILE_USERS

Întreţinerea fişierului cu parole se referă la executarea activităţilor de *extindere, relocare, ştergere sau schimbare a stării* acestui fişier.

4.2. CREAREA, PORNIREA ŞI OPRIREA UNEI BAZE DE DATE ORACLE

Configurarea serverului Oracle presupune următoarele activităţi:

- Instalarea sistemului Oracle, care constă în instalarea nucleului Oracle pe server, a instrumentelor (tools-urilor) de aplicaţie pe staţii;
- Evaluarea resurselor fizice ale calculatorului pe care se va instala serverul Oracle şi baza de date;
- Definirea structurii logice a bazei de date şi a strategiei de salvare (backup);
- Crearea şi deschiderea bazei de date;
- Implementarea bazei de date proiectate, prin definirea segmentelor de revenire (rollback), a tabelelor spaţiu şi a obiectelor bazei de date;
- Salvarea bazei de date şi definirea utilizatorilor bazei de date, în concordanţă cu licenţa Oracle.

Pentru a crea o bază de date Oracle trebuie să avem suficientă memorie pentru pornirea unei instanţe Oracle şi pentru crearea tuturor obiectelor proiectate ale bazei de date. Dacă la momentul instalării s-a creat şi o bază de date iniţială atunci aceasta poate fi dezvoltată astfel încât să cuprindă, în final, toate obiectele bazei de date proiectate. De asemenea această bază de date iniţială poate fi ştearsă şi în locul ei să se creeze o nouă bază de date. Dacă am folosit o versiune anterioară Oracle se poate crea o bază de date nouă în întregime, dacă nu ne mai interesează vechea bază de date, altfel putem migra această bază de date la noua versiune Oracle.

Crearea unei baze de date se face în următorii paşi:

- Salvarea completă a bazei de date existente;
- Crearea *fişierului cu parametrii* folosit la pornirea bazei de date.
- Editarea noului fişier cu parametrii, astfel încât parametrii acestuia să corespundă cerinţelor noii baze de date.

- Controlul identificatorului instanței Oracle, care trebuie să fie identic cu numele bazei de date setat în parametrul DB_NAME;
- Pornirea utilitarul Entreprice Manager şi conectarea la Oracle ca administrator.
- Pornirea unei *instanțe Oracle* (System Global Area şi procesele background) cu opțiunea Startup Nomount.
- Crearea noii bazei de date folosind comanda SQL *CREATE DATABASE*, prin intermediul căreia Oracle execută: crearea fişierelor de date (data files), fişierelor de control (control files) şi a fişierelor de refacere (redo log) ale bazei de date; crearea tabelei spațiu SYSTEM şi a segmentului rollback SYSTEM; crearea dicționarului de date; crearea utilizatorilor SYS şi SYSTEM; specifică setul de caractere care va fi folosit la memorarea datelor în baza de date; montează şi deschide baza de date pentru utilizare.
- Salvarea integrală a bazei de date.

Parametrii de inițializare a bazei de date furnizează valorile necesare pentru funcționarea acesteia sub o anumită instanță Oracle. Aceştia se personalizează prin intermediul unui fişier text, numit fişierul cu parametrii de inițializare. Acesta este citit la momentul pornirii bazei de date de către serverul Oracle. Pentru a face eventuale modificări, baza de date trebuie oprită complet şi repornită după ce s-au efectuat astfel de modificări.

Mulți parametrii de inițializare se folosesc pentru ajustarea şi creşterea performanțelor bazei de date. Specificarea parametrilor se realizează după următoarele reguli:

- Toți parametrii sunt opționali;
- În fişierul cu parametrii se vor fi introduce numai parametrii şi comentarii;
- Semnul (#) marchează începutul unui comentariu, restul liniei fiind ignorat;
- Serverul Oracle are valori asumate pentru fiecare parametru;
- Parametrii pot fi specificați în orice ordine;
- Fişierul de parametrii nu este „case-sensitive" ;
- Pentru a introduce mai mulți parametrii pe o linie aceştia se vor separa prin spațiu:
  PROCESSES = 100 SAVEPOINTS = 5 OPEN_CURSORS = 10
- Unii parametrii, ca de exemplu ROLLBACK_SEGMENTS, acceptă valori multiple, acestea putându-se specifica între paranteze şi separate prin virgulă, sau fără paranteze şi virgule, ambele sintaxe fiind valide, astfel:
  ROLLBACK_SEGMENTS = (SEG1, SEG2, SEG3, SEG4, SEG5)

ROLLBACK_SEGMENTS = SEG1 SEG2 SEG3 SEG4 SEG5

- Caracterul (/) foloseşte pentru marcarea întreruperii scrierii unui parametru pe o linie şi continuarea lui pe linia următoare, imediat fără nici un spaţiu în faţă, astfel:

  **ROLLBACK_SEGMENTS = (SEG1, SEG2, **
  SEG3, SEG4, SEG5)

- Parametrul IFILE se poate introduce într-un fişier cu parametrii alt fişier cu parametrii, astfel:

  IFILE = COMMON1.ORA

Se pot folosi trei niveluri de imbricare. În exemplu de mai sus fişierul COMMON1.ORA poate conţine un al doilea parametru IFILE pentru fişierul COMMON2.ORA, care la rândul său poate conţine un al treilea parametru IFILE pentru fişierul COMMON3.ORA. De asemenea, se pot utiliza mai mulţi parametrii IFILE în acelaşi fişier, astfel:

IFILE = DBPARMS.ORA
IFILE = GCPARMS.ORA
IFILE = LOGPARMS.ORA

Dacă valoarea unui parametru conţine caractere speciale, atunci caracterul special trebuie precedat de caracterul de comutare (\) sau întreaga valoare este inclusă între caracterele (“ ”), ca în exemplul de mai jos:

DB_DOMAIN = JAPAN.ACME\#.COM
sau
DB_DOMAIN = "JAPAN.ACME#.COM"

Pentru schimbarea valorii unui parametru, aceasta se editează în fişierul cu parametrii. Când instanţa Oracle este repornită aceasta va folosi noua valoare a parametrului.

Câţiva parametrii de iniţializare sunt dinamici, în sensul că valorile acestora se pot modifica prin procedeul de mai sus sau în timp ce o instanţă rulează, folosind comenzile SQL: ALTER SESSION, ALTER SYSTEM sau ALTER SYSTEM DEFERRED cu sintaxele:

ALTER SESSION SET nume_parametru = valoare
ALTER SYSTEM SET nume_parametru = valoare
ALTER SYSTEM SET nume_parametru = valoare DEFERRED

Comanda ALTER SESSION schimbă valoarea unui parametru numai la nivelul sesiunii care a lansat-o, după repornirea bazei de date se va utiliza iarăşi valoarea din fişierul cu parametrii.

Comanda ALTER SYSTEM modifică valoarea globală a parametrului, la nivelul întregului sistem, deci pentru toate sesiunile active, după repornirea bazei de date se va utiliza iarăşi valoarea din fişierul cu parametrii.

Comanda ALTER SYSTEM DEFERRED modifică valoarea globală a parametrului nu pentru sesiunile active, ci pentru sesiunile viitoare, care vor fi active după repornirea bazei de date.

Afişarea valorilor curente ale parametrilor de inițializare ai bazei de date se face cu comanda SHOW PARAMETERS, afişarea făcându-se în ordinea alfabetică a parametrilor. Listarea la imprimantă a parametrilor afişați se face cu comanda SPOOL.

Parametrii de inițializare se pot grupa în:

- *Parametrii derivați* sunt cei ale căror valori se calculează pornind de la valorile altor parametrii. Normal valorile acestora nu trebuie modificate;
- *Parametrii globali prefixați cu GC* sunt folosiți pe sistemele care suportă Oracle Parallel Server;
- *Parametrii dependenți de sistemul de operare* sunt cei ale căror valori sunt dependente de specificul sistemului de operare gazdă. Exemplu: DB_BLOCK_BUFFERS care indică numărul ariilor de date (data buffers) din memoria principală sau DB_BLOCK_SIZE care indică mărimea unui bloc de date;
- *Parametrii de tip variabilă* sunt cei ce pot lua anumite valori care să determine performanțele sistemului sau anumite limite de funcționare. Exemplu: OPEN_CURSORS dacă i se dă valoarea 10 se vor putea deschide maxim 10 cursoare, iar DB_BLOCK_BUFFERS prin valorile pe care le va lua nu va impune anumite limite dar va modifica performanțele sistemului;
- *Parametrii statici* sunt cei ale căror valori nu se pot modifica în timpul unei sesiuni sau cu baza de date pornită;
- *Parametrii dinamici* sunt cei ale căror valori se pot modifica, aşa cum s-a arătat mai sus cu comenzile ALTER SESSION, ALTER SYSTEM sau ALTER SYSTEM DEFERRED;

Aşa cum s-a prezentat mai sus compania Oracle furnizează un fişier inițial cu parametrii cu ajutorul căruia putem să pornim o bază de date Pentru a personaliza baza de date conform cerințelor beneficiarului, administratorul va trebui să seteze valori noi pentru anumiți parametrii specificați mai jos, astfel:

- **DB_NAME** *defineşte numele bazei de date*. Se formează dintr-un şir de maximum opt caractere. Dacă nu se furnizează, Oracle atribuie bazei de date un nume standard. Acesta se găseşte în fişierul cu parametrii furnizat o dată cu software-ul Oracle. În timpul creerii bazei de date numele acesteia este scris în fişierele *de date, de control şi de log*.

Exemplu: DB_NAME = BAZA1

- **DB_DOMAIN** este format dintr-un şir de caractere şi defineşte domeniul din rețea căruia aparține baza de date, de obicei este definit de numele organizației căreia aparține baza de date. Dacă se utilizează numele domeniului din Internet, atunci partea adresei de e-mail care urmează după caracterul ”@” este foarte bună pentru a fi folosită ca valoare pentru acest parametru.
  Exemplu: DB_DOMAIN = BUC.ORG.COM
- **GLOBAL_NAMES** defineşte numele global al bazei de date în cadrul rețelei de calculatoare şi este format din numele bazei de date şi numele domeniului separate prin punct. Acest parametru mai poate lua şi valorile: TRUE (caz în care se forțează ca numele global al bazei de date să fie identic cu cel al bazei de date) sau FALSE (numele global nu are semnificație, nu se utilizează).
  Exemplu: GLOBAL_NAMES =FALSE
- **CONTROL_FILES** defineşte numele fişierelor de control ce vor fi create pentru baza de date(se va furniza pentru fiecare fişier calea completă de acces la acesta). Este recomandat ca să se definească cel puțin două fişiere de control, care să fie plasate pe două discuri diferite.
  Exemplu: CONTROL_FILES =diska:cntrl1.ora,diskb:cntrl2.ora
- **LOG_FILES** specifică numărul maxim de grupuri de fişiere de log ce pot fi utilizate pentru o bază de date. Ia valori de la 2 la 255. Acest parametru specifică totodată şi numărul minim de fişiere de log ce pot fi deschise pentru o bază de date.
- **DB_FILE_MULTIBLOCK_READ_COUNT** defineşte *numărul de blocuri* citite simultan pentru accesarea unei tabele a bazei de date. Este folosit pentru optimizarea parcurgerii totale a unei tabele atunci când se caută o anumită valoare a unei coloane aferentă unui rând din aceasta.
- **REMOTE_LOGIN_PASSWORDFILE** specifică dacă se foloseşte sau nu fişierul cu parole pentru identificarea utilizatorilor ce pot executa activități de administrator. Poate lua valorile: NONE nu se foloseşte fişierul de parole iar utilizatorul care va executa activități de administrare trebuie să fie autentificat de către sistemul de operare gazdă; EXCLUSIVE se foloseşte un singur fişier cu parole pentru o singură bază de date. Pe lângă utilizatorii SYS şi SYSTEM şi alți utilizatori pot executa sarcini de administrare; SHARED se foloseşte un singur fişier cu parole pentru mai multe baze de date, caz în care singurii utilizatori ce pot executa activități de administrare sunt SYS şi SYSTEM. În acest caz nu se pot adăuga alți utilizatori în fişierul cu parole.

- **DB_FILES** specifică numărul maxim de fişiere de date ce pot fi deschise de către o bază de date. De fiecare dată când se modifică acest parametru baza de date se opreşte şi apoi se reporneşte.
- **LOG_CHECKPOINT_INTERVAL** specifică frecvenţa punctelor de control (checkpoites) ;
- **LOG_CHECKPOINT_TIMEOUT** specifică timpul maxim dintre două puncte de control în secunde;
- **PROCESSES** specifică numărul maxim de utilizatori care se pot conecta simultan la baza de date, iar acest număr trebuie să fie mai mare decât numărul total de procese background, Job Queue şi Parallel Query;
- **ROLLBACK_SEGMENTS** defineşte toate segmentele rollback pe care o instanţă le poate acapara la momentul pornirii. Valoarea acestui parametru se dă sub forma unei liste de valori.
  Exemplu:
  ROLLBACK_SEGMENTS = (rbseg1, rbseg2, rbseg3, rbseg4)
- **LICENSE_MAX_SESSIONS** specifică numărul maxim de utilizatori concurenţiali ce pot fi admişi în timpul unei sesiuni;
- **LICENSE_MAX_USERS** specifică numărul maxim de utilizatori ce pot fi creaţi pentru o bază de date. LICENSE_MAX_SESSIONS şi LICENSE_MAX_USERS nu pot avea simultan valori diferite de zero, deci unul din aceşti parametrii trebuie să fie setat pe zero;
- **LICENSE_SESSIONS_WARNING** specifică numărul de utilizatori concurenţiali. Dacă se depăşeşte, se emite un mesaj de atenţionare;
- **TRANSACTIONS_PER_ROLLBACK_SEGMENT** specifică numărul tranzacţiilor concurente permise pe un segment rollback;
- **AUDIT_TRAIL** activează sau dezactivează scrierea rândurilor în fişierul de audit. Înregistrările de audit nu se scriu dacă parametrul are valoarea NONE sau lipseşte.

Zona de memorie SGA (System Global Area) conţine următoarele structuri de memorie: *database buffer cache, redo log buffer şi shared pool. Database Buffer Cache* este porţiunea din SGA ce conţine blocurile de date citite din fişierele de date ale bazei de date. *Redo log buffer* este zona în care se păstrează informaţii despre modificările efectuate în baza de date. *Shared pool* este o zonă care conţine la rândul său trei structuri de memorie*: library cache, dictionary cache şi control structures.* Dicţionarul de date al bazei de date este citit în zonele library cache şi dictionary cache.

Mărimea **SGA** este influenţată de valorile parametrilor următori:

- **DB_BLOCK_SIZE** defineşte, în bytes, mărimea unui singur bloc de date. Valorile tipice pentru acest parametru sunt 2048 şi 4096;

- **DB_BLOCK_BUFFERS** specifică numărul de buffere ale bazei de date disponibile în zona *database buffer cache,* a cărei mărime este egală cu DB_BLOCK_SIZE * DB_BLOCK_BUFFERS;
- **LOG_BUFFER** determină mărimea în bytes a zonei *redo log buffer*
- **SHARED_POOL_SIZE** specifică mărimea în baiți a ariei *Shared pool.* Acest parametru poate accepta valori numerice urmate de literele "K" sau "M", unde "K" înseamnă că numărul va fi multiplicat cu 1000, iar "M" înseamnă că numărul va fi multiplicat cu 1000000.
- **OPEN_CURSORS** specifică numărul maxim de cursoare pe care o sesiune le poate deschide simultan. Valoarea asumată este de 50. Este bine ca acest număr să fie cît mai mare pentru a nu avea probleme cu rularea unei aplicații.
- **TRANSACTIONS** specifică numărul maxim de tranzacții concurente. O valoare mare a acestui parametru determină mărirea zonei de memorie SGA.

Exemple de creare a unei baze de date

1) ***CREATE DATABASE;***

În acest caz se crează o bază de date Oracle în care toți parametrii comenzii CREATE DATABASE iau valorile standard furnizate de firma Oracle.

2) ***CREATE DATABASE baza1***

```
   LOGFILE   GROUP 1 ('diskb:log1.log', 'diskc:log1.log') SIZE 50K,
             GROUP 2 ('diskb:log2.log', 'diskc:log2.log') SIZE 50K
   MAXLOGFILES 5
   MAXLOGHISTORY 100
   MAXDATAFILES 10
   ARCHIVELOG
        CHARACTER SET US7ASCII
        DATAFILE 'diska:datfile1.dat' SIZE 2M
        DATAFILE
              'disk1:datfile2.dbf'   AUTOEXTEND   ON
              'disk2:datfile3.dbf'   AUTOEXTEND   ON   NEXT   10M
      MAXSIZE UNLIMITED;
```

După crearea unei baze de date, instanța Oracle poate fi lăsată să ruleze, iar baza de date este deschisă şi montată pentru utilizare normală. Pentru opririle şi pornirile ulterioare se poate utiliza *Oracle Enterprise Manager*.

Pornirea şi oprirea bazei de date

O bază de date şi instanţa Oracle se poate porni cu utilitarul *Oracle Enterprise Manager* folosind fereastra de dialog Startup Database. O *instanţă şi o bază de date* asociată se pot porni în mai multe moduri.

Pornirea instanţei *fără montarea* bazei de date se face atunci când dorim să creăm o bază de date. Activitatea se execută din fereastra de dialog Startup Database prin selectarea butonului radio Startup Nomount;

Pornirea instanţei şi *montarea bazei de date,* aceasta rămânând *închisă* se execută atunci când dorim să executăm anumite activităţi de întreţinere: redenumirea fişierelor de date; adăugarea, ştergerea sau redenumirea fişierelor redo log; recuperarea integrală a bazei de date; Această pornire se execută din fereastra de dialog Startup Database prin selectarea butonului radio Startup Mount. Montarea bazei de date se poate execută şi după pornirea unei instanţe fără bază de date montată, cu ajutorul comenzii SQL *ALTER DATABASE* cu opţiunea *MOUNT.*

Exemplu: SQL> ALTER DATABASE baza1 MOUNT;

Pornirea instanţei, *montarea bazei de date* şi *deschiderea acesteia* se face în mod *nerestricţionat* (accesibilă tuturor utilizatorilor care au cu privilegiul CREATE SESSION) sau *restricţionat* (accesibilă doar utilizatorilor de tip DBA, utilizatorilor cu privilegiile CREATE SESSION şi RESTRICTED SESSION).

În modul de pornire restricţionat se pot executa activităţi ca: recrearea indecşilor; exportul sau importul datelor bazei de date; încărcarea datelor cu utilitarul SQL*Loader; blocarea temporară a accesului utilizatorilor obişnuiţi la baza de date. *Pornirea în mod nerestricţionat* se face din fereastra de dialog Startup Database prin selectarea butonului radio Startup Open. *Pornirea în mod restricţionat* se face din fereastra de dialog Startup Database prin selectarea butonului radio Restrict

Deschiderea unei baze de date se poate face după ce instanţă a fost pornită, baza de date montată dar închisă, cu ajutorul comenzii SQL *ALTER DATABASE* cu opţiunea *OPEN.*

Exemplu: SQL> ALTER DATABASE baza1 OPEN;

Transformarea modului de pornire normală a unei baze de date în modul restricţionat se poate face şi cu comanda SQL *ALTER SYSTEM* cu opţiunea *ENABLE RESTRICTED SESSION ,* iar revenirea la situaţia iniţială se face cu aceeaşi comandă dar cu opţiune*a DISABLE RESTRICTED SESSION*

Exemplu:

SQL> ALTER SYSTEM ENABLE RESTRICTED SESSION;
SQL> ALTER SYSTEM DISABLE RESTRICTED SESSION;

În anumite circumstanțe este posibil ca activitățile de pornire a bazei de date şi instanței Oracle să se execute altfel decât în mod uzual. Astfel putem avea:

- *pornirea forțată a unei instanțe,* care se poate realiza atunci când instanța curentă nu poate fi oprită cu succes prin folosirea butoanelor radio Normal sau Immediate din fereastra de dialog Startup. În acest caz se poate forța pornirea unei noi instanțe Oracle, care va determina oprirea instanței anterioare aflată în situația de mai sus.
- *pornirea unei instanțe, montarea bazei de date şi pornirea procesului de* ***recuperare*** *a bazei de date, a tabelelor spațiu sau a fişierelor de date,* care se execută atunci când ştim că mediul bazei de date are nevoie de recuperare.
- *pornirea în modul exclusiv sau paralel,* care se face atunci când avem un server Oracle care permite accesul mai multor instanțe la aceeaşi bază de date.
- *pornirea automată a bazei de date la momentul pornirii sistemului de operare*, se face dacă dorim acest lucru.
- *pornirea unei instanțe şi a unei baze de date la distanță,* se face atunci când serverul Oracle este o parte a unui sistem de baze de date distribuite.

Oprirea unei baze de date se poate face în două moduri: normal sau forțat.

Modul normal, în care oprirea bazei de date se face ca revers al operației de pornire normală, sens în care se execută închiderea bazei de date, demontarea bazei de date şi oprirea instanței Oracle. Activitatea se execută din fereastra de dialog Shutdown Database prin selectarea butonului radio Normal. Oprirea unei baze de date în condiții normale presupune executarea de către Oracle a următoarelor activități: oprirea conectărilor la baza de date; deconectarea tuturor utilizatorilor; la următoarea pornire a bazei de date nu se pornesc procedurile de recuperare.

Modul forțată se poate execută în două moduri: imediat sau prin anularea instanței.

Oprirea imediată *a bazei de date* se execută în cazul unui incident iminent. În cadrul acestei opriri Oracle execută instrucțiunea SQL aflată în lucru şi orice altă tranzacție nefinalizată este anulată prin procesul de rollback; toți utilizatorii conectați sunt deconectați imediat.

Oprirea se face din fereastra de dialog Shutdown Database prin selectarea butonului Immediate.

Oprirea prin anularea instanței se execută dacă baza de date sau una din aplicațiile sale funcționează anormal şi nici una din metodele de oprire anterioare nu funcționează. Această oprire se execută din fereastra de dialog Shutdown prin setarea butonului radio Abort.

În timpul acestei opriri Oracle finalizează instrucțiunile SQL aflate în lucru, tranzacțiile nefinalizate nu mai sunt aduse la starea anterioară momentului începerii acestora (nu mai sunt anulate prin procesul de roll back), iar toți utilizatorii sunt deconectați imediat.

4.3. CREAREA ŞI ACTUALIZAREA TABELELOR

Comanda de creare a unei tabele, **CREATE TABLE**, realizează fie definirea unei tabele, urmând ca introducerea de date să se efectueze ulterior, fie definirea şi încărcarea cu date a unei tabele, chiar în momentul creerii ei, folosind sintaxele:

CREATE TABLE nume-tabelă
(spec-col [NOT NULL],...)
SPACE definire-spațiu [PCTFREE n] |
CLUSTER nume-cluster (nume-col,...)];

CREATE TABLE nume-tabelă
[(nume-col [NOT NULL],...)]
[SPACE definire-spatiu [PCTFREE n]
CLUSTER nume-cluster (nume-col,...)]
[AS cerere]

Unde:
spec-col cuprinde **nume-col**, format, mărime.
nume-col este numele coloanei din tabelă.
format reprezintă formatul coloanei din tabelă.Formatele acceptate sunt specificate în tabelul următor:

Tipul datelor	Formatul	Explicații
Char	CHAR(mărime)	Date alfanumerice. Marimea maximă a coloanei este de 240 caractere.
Date	DATE	Permit introducerea câmpurilor de tip dată. Exemplu: January 1, 4712 BC to December 31, 4712 AD

Long	LONG	Date caracter de mărime variabilă, într-o tabelă trebuie să fie definită doar o coloană de tip LONG.
	LONG VARCHAR	La fel ca LONG.
	LONG ROW	Date binare RAW.
Numerice	NUMBER	Date numerice de mărime implicită.
	NUMBER (mărime)	Date numerice de mărime specificată.
	NUMBER (întregi zecimali)	Date numerice în baza zece.
	NUMBER(*)	La fel ca datele de tip NUMBER.
	DECIMAL	La fel ca datele de tip NUMBER.
	FLOAT	Nu acceptă descrierea mărimii coloanei.
	INTEGER	La fel ca datele de tip NUMBER.
	INTEGER (mărime)	Nu acceptă descrierea mărimii zecimalelor.
	SMALLINT	La fel ca date de tip INTEGER
RAW	RAW(mărimea)	Date binare RAW. Mărimea maximă este 240 octeți.
	LONGRAW	Date de tip LONG.

În cazul celei de-a doua sintaxe, tipul şi mărimea coloanelor vor fi copiate din rezultatul obținut în urma specificării **AS** cerere. În ambele sintaxe se utilizează alte cuvinte cheie sau parametrii, care au următoarea semnificație:

NULL sau **NOT NULL** se referă la câmpurile din coloane care pot avea sau nu valori nule. Clauzele nume-col [**NOT NULL**] se pot repeta pentru maximum 254 de coloane.

SPACES desemnează modelul de alocare a spațiului pentru tabela ce se crează. Modelul este creat în prealabil cu **CREATE SPACES**. Dacă parametrul **SPACE** este omis, pentru toți parametrii de spațiu vor fi utilizate valorile implicite din comanda **CREATE SPACE**.

PCTFREE permite specificarea unei alte valori dectt cea implicită de 20% pentru spațiul lăsat liber. Are aceeaşi funcție ca şi în comanda **CREATE SPACE.**

CLUSTER este parametrul care indică faptul câ tabela va fi inclusă în clusterul cu numele nume-cluster. Introducerea datelor în tabela va declanşa memorarea lor în ordinea indicelui de cluster.

Nume-col se referă la numele coloanelor din tabelă care corespund coloanelor din cluster.

Exemplu:

Să se creeze tabelele: PRODUSE, CLIENȚI, PRETURI SALARIATI, DEPOZITE, COMENZI ce intră în componența bazei de date COMBAZA.

```
SQL> CREATE TABLE PRODUSE
2 (CODD NUMBER(6) NOT NULL,
3 CODP NUMBER(5) NOT NULL,
4 DENP CHAR(11),
5 STOC NUMBER(6),
6 DATACRT DATE,
7 UM CHAR(3));
 Table created.
SQL> CREATE TABLE CLIENȚI
2 (CODC NUMBER(6) NOT NULL,
3 DENC CHAR(11),
4 LOC CHAR(11),
5 STR CHAR(16),
6 NRCHARC3),
7 CONTNUMBER(11),
8 TEL NUMBER(8),
9 TFAX NUMBER(8))
Table created.
SQL> CREATE TABLE PRETURI
2 CODP NUMBER(5) NOT NULL,
3 PRETMAX NUMBEB(9,2),
4 PRETMIN NUMBER(9,2), 6 DATAI DATE,
6 DATASF DATE);
Table created.
SQL> CREATE TABLE SALARIATI
2 (MARCA NUMBER(4) NOT NULL,
3 NUME CHAR(16),
4 FUNCT CHAR(7),
6 CODD NUMBER(G) NOT NULL, 6 SALA NUMBER(9,2)
7 VENS NUMBER(9.2),
8 CODS NUMBER(4) NOT NULL);
Table created.
SQL> CREATE TABLE DEPOZITE
2 (CODD NUMBER(6) NOT NULL,
3 DEND NUMBER(6) NOT NULL,
4 NRSAL NUMBER(2));
Table created.
SQL> CREATE TABLE COMENZI
2 (NRCOM NUMBER(6) NOT NULL,
```

3 CODP NUMBER(5) NOT NULL,
4 CODC NUMBEIK6) NOT NULL,
5 DATAL DATE,
6 CANT NUMBER(7),
7 PREȚ NUMBER(9-2));
Table created.

Modificarea unei tabele presupune adăugarea de noi coloane la sfîrşitul acesteia sau modificarea tipurilor unor coloane deja existente. Comanda care realizează aceste operații este ALTER TABLE. Sintaxa ei este:

ALTER TABLE table {ADD | MODIFY}
(spec-col [NULL | NOT NULL,...])

unde:
ADD adaugă la sfârşitul tabelei noi coloane care inițial au valori nule; ulterior, prin actualizare, aceste coloane se vor completa cu date.
MODIFY schimbă definirea unei coloane existente. Modificarea tipului sau mărimii unei coloane presupune ca aceasta să conțină numai valori nule.
spec-col reprezintă numele, formatul şi mărimea coloanei ce se va adăuga sau se va modifica.
NULL se referă la faptul că valorile din coloană ce se adaugă/modifică sunt nule.
NOT NULL specifică faptul că o coloană nu poate avea valori nule (**NOT NULL**). Unei coloane deja existente i se poate ataşa parametrul **NOT NULL** numai dacă aceasta conține valori nenule.

Exemple:
1) Să se adauge la tabela DEPOZITE o nouă coloană CANTDEP ce nu a fost prevăzută inițial.
SQL> ALTER TABLE DEPOZITE ADD
2 (CANTDEP NUMBER(7));
Table altered.
2) Să se modifice atributele coloanei CANTDEP (respectiv mărirea acesteia cu două unități).
SQL> ALTER TABLE DEPOZITE MODIFY
2 (CANTDEP NUMBER(9));
Table altered.

Pentru ştergerea unei tabele într-o bază de date se utilizează comanda DROP TABLE. Sintaxa ei este:

DROP TABLE nume-tabelă:

În general, nu se pot şterge tabelele create de alţi utilizatori. La ştergerea unei tabele. se şterg automat indecşii corespunzători (create fie de către proprietarul tabelei, fiu de către alţi utilizatori) şi privilegiile conferite în legătură cu ea. Rămân, însă, viziunile şi sinonimele referitoare la tabela ştearsă, care vor deveni invalide. Va trebui fie ca acestea să fie şterse, fie să se definească sau să se redefinească tabela în aşa fel incât viziunile şi sinonimele să devină valide.

Exemplu:

Să se steargă tabela DEPOZITE.

SQL> DROP TABLE DEPOZITE;
Table dropped.

Se pot modifica numele atribuite tabelelor, sau sinonimelor utilizând comanda **RENAME**. Sintaxa ei este:

RENAME nume-vechi TO nume-nou;

unde:

nume-vechi şi **nume-nou** sunt constante de tip şir de caractere (nu se scriu între ghilimele sau apostrofuri).

4.4. CREAREA ŞI ACTUALIZAREA INDECŞILOR

În vederea obţinerii de performanţe superioare privind accesul la tabelele unei baze de date se pot construi indecşi folosind comanda CREATE INDEX.

Comanda are ca efect crearea unui index la o tabelă, index care va conţine câte o intrare pentru fiecare valoare care apare în coloana specificată pentru tabela respectivă. Această coloană se numeşte coloana de index.

Sintaxa comenzii **CREATE INDEX** este:

CREATE [UNIQUE] INDEX nume-index
ON nume-tabela (nume-col [ASC | DESC], nume-col [ASC | DESC],...)
[COMPRESS | NOCOMPRESS]
[SYSSORT | NOSYSSORT]
[ROWS =n][PCTFREE = {20 | n}];

unde:
UNIQUE se referă la faptul că tabela nu va conţine două rânduri cu aceleaşi valori în coloanele index (deci nu vor exista dubluri în coloana index). Dacă se omite, tabela poate conţine oricâte asemenea rânduri care să aibă în coloana index aceeaşi valoare.
ASC|DESC precizează ordinea în care va fi păstrat indexul: ascendentă sau descendentă.
COMPRESS indică faptul că indecşii pot fi comprimaţi, reducându-se în acest mod spaţiul necesar memorării acestora şi mărindu-se viteza operaţiilor care folosesc indecşii. Comprimarea nu permite găsirea într-un index a valorii unui anumit câmp fără să acceseze tabela. **NOCOMPRESS** suprimă comprimarea indecşilor.
SYSSORT specifică faptul că procedura standard ORACLE de sortare este folosită pentru a crea un index. **NOSYSSORT** este folosit pentru depanare.
ROWS specifică numărul aproximativ de rânduri ce urmează a fi indexate. Acest număr este utilizat pentru a optimiza sortarea în **SYSSORT**. Clauza nu are efect în **SQL*Plus**.
PCTFREE specifică, în momentul creării indexu-lui, procentul din spaţiul pentru index ce trebuie să rămînă liber pentru actualizări. Nu are efect asupra extensiilor alocate după crearea indexului.

Indexul creat permite creşterea vitezei de acces la datele unei tabele prin intermediul coloanei index datorită faptului că se vor căuta rândurile corespunzătoare valorilor coloanei index, fără a citi întreaga tabelă. Deoarece indexul şi tabela trebuie actualizate, introducerile, ştergerile şi modificările în câmpul de valori ale coloanei index necesită un timp mai mare. De aceea, se recomandă crearea de maxim 3 indecşi pentru o tabelă. In funcţie de necesităţi, se pot creea şi şterge în mod dinamic indecşi.

Indecşii creaţi pentru o singură tabelă trebuie să aibă nume distincte. Pentru tabele diferite pot fi creaţi indecşi cu acelaşi nume. În momentul ştergerii acestor indecşi trebuie specificată tabela din care fac parte.

Comanda de validare a unui index este **VALIDATE INDEX**. Această comandă verifică integritatea indexului specificat pentru o anumită tabelă. Dacă pe tabela respectivă a fost creat un singur index, al cărui nume va fi specificat în comandă, atunci numele tabelei poate fi omis. Comanda **VALIDATE INDEX** are sintaxa:

VALIDATE INDEX nume-index
[ON nume-tabelă] [WITH LIST];

unde:

WITH LIST salvează informațiile referitoare la index într-un fişier.
nume-index este numele indexului ce urmează a fi validat. Dacă indexul este valid, comanda va afişa un mesaj corespunzător (Index validated). Obținerea oricărui alt mesaj va determina ştergerea indexului şi recrearea lui.
Ştergerea unui index din baza de date se realizează cu ajutorul comenzii **DROP INDEX**, cu sintaxa:

DROP INDEX nume-index [ON nume-tabelă];

Dacă pe o anumită tabelă a fost creat un singur index, numele tabelei respective poate fi omis în linia de comandă. Este posibil ca doi indecşi creați pe tabele diferite să aibă acelaşi nume. În acest caz, în linia de comandă vor apărea atît numele indexului cât şi al tabelei pentru care a fost creat. Se recomandă numai ştergerea indecşilor proprii.
Exemple:
1) Să se creeze un index cu numele SUMAR pentru coloana FUNCT din tabela SALARIAȚI.

SQL> CREATE INDEX SUMAR
2 ON SALARIATI(FUNCT);
Index created

2) Să se creeze un index cu numele MARCA pentru coloana MARCA din tabela SALARIAȚI.

SQL> CREATE INDEX MARCA
2 ON SALARIAT(MARCA);
Index created.

3) Să se creeze un index cu numele MARCA din pentru coloana MARCA clia tabela SALARIAȚI cu suprimarea comprimării lui.

SQL> CREATE INDEX MARCA
2 ON SALARIATI(MARCA) NOCOMPRESS;
Index created.

4) Să se verifice integritatea indexului SUMAR.

SQL> VALIDATE INDEX SUMAR;
Index validated.

6) Să se verifice integritatea indexului MARCA pentru tabela SALARIAȚI.

SQL> VALIDATE INDEX MARCA ON SALARIAȚI;
Index validated.

7) Să se şteargă indexul SUMAR.

SQL> DROP INDEX SUMAR;
Index dropped.

8) Să se şteargă indexul MARCA din tabela SALARIAȚI.

SQL> DROP INDEX MARCA ON SALARIAȚI;

Index dropped.

4.5. CREAREA ŞI ACTUALIZAREA TABELELOR ŞI INDECŞILOR PARTIŢIONAŢI

Partiţionarea este procesul de împărţire a tabelelor şi indecşilor foarte mari în mai multe tabele, şi respectiv indecşi, mai mici numite ***partiţii*** pentru a le putea manipula mai uşor.

O dată definite partiţiile, instrucţiunile SQL pot accesa şi manipula aceste partiţii în loc de a manipula întreaga tabelă sau întregul index din care au provenit.

Toate partiţiile au aceleaşi atribute logice ca şi tabela sau indexul din care au provenit. De exemplu toate partiţiile unei tabele au aceleaşi coloane cu aceleaşi constrângeri de integritate ca şi tabela, iar partiţiile unui index sunt construite după aceeaşi coloană ca şi indexul din care au provenit.

Fiecare partiţie este memorată într-un segment propriu aflat în aceeaşi tabelă spaţiu sau în tabele spaţiu diferite. Plasarea partiţiilor în tabele spaţiu diferită prezintă următoarele avantaje semnificative:

- Riscul pierderii datelor poate fi diminuat;
- Salvarea şi restaurarea partiţiilor se poate face independent unele faţă de altele;
- Se poate realiza o echilibrare a operaţiilor de I/O prin maparea partiţiilor pe discuri diferite.

O partiţie este definită de următoarele:

- *Numele partiţiei* identifică în mod unic partiţia în schema de obiecte a unui utilizator;
- *Referirea unei partiţii* se face în context obligatoriu cu numele tabelei sau indexului din care provine. Numele unei partiţii poate să apară în instrucţiuni DDL şi DML, precum şi în utilitarele Import/Export şi SQL*Loader.

  Exemplu:

  ALTER TABLE tab10 DROP PARTITION part1;
- *Aria de cuprindere a partiţiei* este formată din rândurile tabelei sau indexului, care aparţin acesteia bazat pe valorile coloanele ce definesc partiţia. Aria de cuprindere este definită prin clauzele:

  PARTITION BY RANGE (coloana1, colana2, …)

  şi

  VALUES LESS THAN (valoare1, valoare2, …)

unde *coloana1, coloana2, ...* formează lista coloanelor. În funcție de valorile acestora se stabilesc rândurile din tabelă sau index care aparțin partiției. Coloanele după care se face partiționarea nu trebuie să conțină valori de tip NULL, iar numărul lor nu poate fi mai mare de cât 16; *valoare1, valoare2, ...* formează lista valorilor coloanelor. În funcție ele se selectează rândurile pentru a fi incluse în partiție. Aceste valori formează limita superioară a partiției. *Limitele partiției* definesc mulțimea rândurilor tabelei sau indexului ce vor fi incluse în aceasta. Orice partiție are două limite, *limita inferioară* a cărei valoare este definită de valoarea LESS THAN din definirea partiției precedente şi **care este inclusă în partiție**, şi *limita superioară este* definită de valoarea LESS THAN din definirea partiției curente, valoare **care nu este inclusă în partiție.** De la această regulă face excepție prima partiție care nu are limită inferioară. Partiționarea nu se poate face după pseudocoloanele LEVEL, ROWID sau MISLABEL. Ca valoare în clauza LESS THAN a ultimei partiții se poate specifica şi MAXVALUE, care reprezintă o valoare virtuală egală cu infinit. *Cheia de partiționare* este un set de valori format din valorile coloanelor de partiționare aferente unui rând al partiției. Specificarea unei valorii alta decât MAXVALUE pentru parametrul LESS THAN *impune o constrângere implicită* de tip CHECK la nivel de tabelă, cu această valoare.

Reguli de partiționare a tabelelor si a indecşilor

Partiționarea tabelelor se face respectând câteva reguli esențiale, astfel:

- O tabelă *poate fi* partiționată dacă *nu* este inclusă într-un grup de tabele sau *nu* conține tipurile de date LOB, LONG, LONG RAW sau obiect;
- O tabelă partiționată sau nepartiționată *poate avea* indecşi şi partiționați şi/sau nepartiționați;
- *Atributele fizice* ale unei partiții pot fi specificate inițial prin comanda CREATE TABLE pentru crearea partiției *implicit* sau *explicit*. *Specificarea implicită* se face prin furnizarea atributelor fizice pentru tabelă şi nespecificarea acestora pentru partiție în clauza PARTITION. Atributele fizice implicite pot fi ulterior modificate cu comanda **ALTER TABLE MODIFY DEFAULT ATTRIBUTES.** *Specificarea explicită* se face prin furnizarea acestora în clauza PARTITION, caz în care valorile atributelor fizice ale partiției vor suprascrie valorile atributelor fizice implicite ale tabelei. Atributele fizice explicite pot fi ulterior modificate cu comenzile

  ALTER TABLE MODIFY PARTITION

sau

ALTER TABLE MOVE PARTITION

Partiţionarea indecşilor se face respectând câteva reguli:

- Un index *poate fi* partiţionat dacă *nu* este inclus într-un grup de indecşi sau nu este definit pentru o tabelă inclusă într-un grup de tabele;
- *Indecşii* pot fi de patru categorii: *locali prefixaţi, locali neprefixaţi, globali prefixaţi şi globali neprefixaţi.* Indecşii globali pot fi partiţionaţi sau nepartiţionaţi. *Un index este local* dacă cheile dintr-o partiţie oarecare a acestuia referă rânduri aflate într-o singură partiţie a tabelei pentru care a fost creat. Indexul *local este prefixat* dacă este definit pe un set de coloane dintre care cea mai din stânga este coloana după care a fost partiţionat atât indexul local, cât şi tabela pe care a fost creat acesta. De exemplu, presupunem că avem tabela TAB10 şi indexul său INDEX10, care au fost partiţionaţi după coloanele COL1 şi CIOL2. Dacă indexul INDEX10 este definit pe coloanele (COL1, COL2, COL3), atunci el este *local prefixat* pentru că în partea cea mai din stânga(prefix) a listei coloanelor de indexare se află coloanele COL1 şi COL2 după care s-a făcut partajarea tabelei şi a indexului său. Dacă indexul INDEX10 ar fi definit pe coloanele (COL1, CO23) atunci indexul este *local neprefixat*. Indecşii locali nu pot fi partiţionaţi cu clauza BY RANGE, această clauză se aplică numai pentru indecşii globali. *Un index este global* dacă cheile dintr-o partiţie oarecare a acestuia referă rânduri aflate în mai multe partiţii ale tabelei pentru care a fost creat. La fel ca şi în cazul indexului local putem avea *index global prefixat* şi *index global neprefixaţ*. Indexul global arată faptul că *partiţionarea acestuia este definită de utilizator* şi *nu trebuie să fie echivalentă* cu partiţionarea tabelei pentru care se creează.
- Definirea atributelor fizice ale indecşilor comportă aceleaşi reguli ca la definirea atributelor fizice ale partiţiilor unei tabele, cu deosebirea că aici se folosesc pentru *crearea unui index* comenzile:

  CREATE INDEX

  sau

  ALTER TABLE SPLIT PARTITION,

  iar pentru *modificarea* atributelor fizice comenzile SQL

  ALTER INDEX MODIFY PARTITION

  sau

  ALTER INDEX REBUILD PARTITION.

Crearea partițiilor unei tabele şi ale unui index

Crearea partițiilor este similară cu cea a creării tabelelor sau indecşilor.

Crearea partițiilor unei tabele se face cu comanda **CREATE TABLE** folosind şi clauza **PARTITION**.

Exemplu:

Avem tabela FACTURI ce cuprinde documente din anii 1999 - 2002. Tabela are printre alte coloane si coloanele AN, LUNA, ZI şi dorim să partiționăm tabela în trei partiții, astfel încât partiția 1 să conțină date din anii 1999 şi 2000, partiția 2 date din anul 2001, iar partiția 3 date din anul 2001. Cu comanda de mai jos vom crea cele 3 partiții, astfel:

```
CREATE TABLE facturi
(numar_factura   NUMBER (12),
 nume_furnizor  VARCHAR(25),
an                NUMBER(4),
luna              NUMBER(2),
zi                NUMBER(2))
STORAGE (INITIAL 100K NEXT 50K) LOGGING
PARTITION BY RANGE (an, luna, zi)
(PARTITION p1_1999_2000 VALUES LESS THAN (2001, 13, 32)
                  TABLESPACE tabsp_1 NOLOGGING,
PARTITION p1_2001 VALUES LESS THAN (2002, 13, 32)
                  TABLESPACE tabsp_2 NOLOGGING,
 PARTITION p1_2002 VALUES LESS THAN (2001, 13, 32)
                  TABLESPACE tabsp_3 NOLOGGING);
```

Crearea partițiilor unui index se face cu comanda **CREATE INDEX** folosind şi clauza **PARTITION**. Activitatea comportă unele diferențe de la un tip de index la altul, conform exemplelor de mai jos:

Exemple:

1) Crearea unui *index local prefixat* pentru tabela de mai sus *cu specificarea* partițiilor:

```
CREATE INDEX index_loc_prefix ON facturi (an, luna, zi, număr_factură)
    LOCAL    (PARTITION  p1 TABLESPACE tabsp1,
              PARTITION  p2 TABLESPACE tabsp1,
              PARTITION  p3 TABLESPACE tabsp1);
```

2) Crearea unui *index local prefixat* pentru tabela de mai sus *fără specificarea* partițiilor:

```
CREATE INDEX index_loc_prefix ON facturi (an, luna, zi, număr_factură) LOCAL;
```

3) Crearea unui *index local neprefixat* pentru tabela de mai sus *cu specificarea* partițiilor:

CREATE INDEX index_loc_prefix ON facturi
LOCAL (PARTITION p1 TABLESPACE tabsp1,
PARTITION p2 TABLESPACE tabsp1,
PARTITION p3 TABLESPACE tabsp1);

4) Crearea unui *index local neprefixat* pentru tabela de mai sus *fără specificarea* partițiilor:

CREATE INDEX index_loc_prefix ON facturi (an, lună, zi) LOCAL;

5) Crearea unui *index global prefixat* pentru tabela de mai sus *cu specificarea* numelui partițiilor şi a parametrului GLOBAL:

CREATE INDEX index_loc_prefix ON facturi (an, luna, zi, număr_factură)
GLOBAL PARTITION BY RANGE (an, luna, zi)
(PARTITION p1_1999_2000 VALUES LESS THAN (2001, 13, 32)
TABLESPACE tabsp_1 NOLOGGING,
PARTITION p1_2001 VALUES LESS THAN (2002, 13, 32)
TABLESPACE tabsp_2 NOLOGGING,
PARTITION p1_2002 VALUES LESS THAN (2001, 13, 32)
TABLESPACE tabsp_3 NOLOGGING);

6) Crearea unui *index global prefixat* pentru tabela de mai sus *fără specificarea* numelui partițiilor şi a parametrului GLOBAL:

CREATE INDEX index_loc_prefix ON facturi (an, luna, zi, număr_factură)
PARTITION BY RANGE (an, luna, zi)
(PARTITION VALUES LESS THAN (2001, 13, 32)
TABLESPACE tabsp_1 NOLOGGING,
PARTITION VALUES LESS THAN (2002, 13, 32)
TABLESPACE tabsp_2 NOLOGGING,
PARTITION VALUES LESS THAN (2001, 13, 32)
TABLESPACE tabsp_3 NOLOGGING);

7) Crearea unui *index global prefixat* pentru tabela de mai sus *cu specificarea* numelui partițiilor şi a parametrului GLOBAL, cu un număr de partiții diferit de cel al partițiilor tabelei pe care se crează şi cu alte coloane de partiționare decât cele după care s-a partiționat tabela:

CREATE INDEX index_loc_prefix ON facturi (an,lună, număr_factură)
GLOBAL PARTITION BY RANGE (an, lună)
(PARTITION p1_1999_2001 VALUES LESS THAN (2002, 13)
TABLESPACE tabsp_1 NOLOGGING,

PARTITION p1_2001 VALUES LESS THAN (2003, 13)
TABLESPACE tabsp_2 NOLOGGING);

Intreținerea partițiilor

Întreținere a partițiilor se realizează prin executarea activităților de *modificare*, *mutare, adăugare, distrugere, trunchiere, splitare, reunire a acestora, precum şi schimbarea partițiilor şi reconstruirea partițiilor index.*

- **Modificarea** unei partiții a unei tabele se face cu comanda **ALTER TABLE** cu clauza **MODIFY PARTITION**. Cu această comandă se pot modifica *atributele fizice* ale partiției sau ale partiției indexului aferent.
- **Mutarea partițiilor** unei tabele sau index se face pentru a schimba tabela spațiu în care rezidă acestea din diverse considerente, dintre care cele de obținere a unor performanțe în exploatarea sunt cele mai frecvente. Operația se execută cu comanda **ALTER TABLE** cu opțiunea **MOVE PARTITION.**
  Exemplu:
  ALTER TABLE tab10 MOVE PARTITION part1 TABLESPACE tabsp10 NOLOGGING;
  Mutarea partiției unei tabele care conține date, determină necesitatea recreării tuturor indecşilor locali sau globali ataşați acesteia.
- **Adăugarea unor noi** partiții se poate executa doar pentru o tabelă partiționată sau un index local. Operația se execută cu comanda **ALTER TABLE** cu opțiunea **ADD PARTITION**.
  Exemplu:
  ALTER TABLE tab100 ADD PARTITION part1 VALUES LESS THAN (935);
  O nouă partiție poate fi adăugată doar după partiția cu limita superioară cea mai mare. Dacă dorim să adăugăm o partiție la început, între partițiile existente sau după ultima partiție care are limita superioară egală cu valoarea MAXVALUE, atunci acest lucru se poate realiza prin splitarea unei partiții, în cazul de față prima, una adiacentă cu locul de inserare a noii partiții, şi respectiv ultima partiție. Dacă pentru tabela partiționată căreia îi adăugăm o nouă partiție există un index local, atunci Oracle creează automat o partiție de index pentru noua partiție adăugată.
- **Distrugerea partițiilor unei tabele** se face după anumite reguli funcție de situațiile în care se află partiția, cu ajutorul comenzii **ALTER TABLE** cu opțiunea **DROP PARTITION**. *Distrugerea partiției unei tabele care conține date şi a indexului global* se poate face lăsând

nealteraţi indecşii globali în timpul distrugerii partiției, după care aceştia vor fi recreați sau ştergând toate rândurile partiției cu comanda **DELETE** după care distrugem partiția. Această comandă actualizează indecşii globali. *Distrugerea partiției unei tabele care conține date şi a constrângerilor referențiale de integritate* se poate face dezactivând constrângerile de integritate, distrugând partiția şi apoi reactivând constrângerile de integritate. Ştergerea rândurilor partiției cu comanda **DELETE**, apoi distrugem partiția.

- **Distrugerea partițiilor unui index** se face după anumite reguli funcție de situațiile în care se află partiția, cu ajutorul comenzii **ALTER INDEX** cu opțiunea **DROP PARTITION. O** partiție a unui index local nu poate fi distrusă, ea se distruge implicit atunci când se distruge partiția tabelei căreia îi corespunde. O partiție fără date a unui index global poate fi distrusă. Dacă o partiție a unui index global conține date, atunci prin distrugerea acesteia partiția următoare devine invalidă şi trebuie recreată**;**
- **Trunchierea partițiilor unei tabele** se face cu comanda **ALTER TABLE** cu opțiunea **TRUNCATE PARTITION** şi are ca efect ştergerea tuturor rândurilor de date din această partiție şi a partiției corespunzătoare a indexului local asociat, dacă există. Partițiile indecşilor nu pot fi trunchiate, singură trunchiere posibilă este cea specificată mai sus. Trunchierea unei partiții a unei tabele şi a indecşilor globali asociați şi/sau a constrângerilor de integritate referențială se face după aceleaşi reguli ca şi operația de distrugere.
- **Splitarea partițiilor** unei tabele sau ale unui index se face cu comanda **ALTER TABLE/INDEX** cu opțiunea **SPLIT PARTITION**. O partiție a unei tabele ce conține date, prin splitare toți indecşii asociați devin inutilizabili şi ca atare aceştia trebuie recreați. Numai partiția fără date nu invalidează indecşii. Partiția unui index poate fi splitată numai dacă indexul este global şi nu conține date, căci partiția unui index local se splitează automat atunci când se splitează partiția tabelei căreia îi corespunde.
- **Fuzionarea sau reunirea** partițiilor unei tabele sau ale unui index se face cu ajutorul comenzilor **ALTER TABLE/INDEX** şi cu una din opțiunile **DROP PARTITION** sau **EXCHANGE PARTITION**, pentru că o opțiune explicită de fuzionare nu există. O partiție a unei tabele poate fi reunită cu altă partiție numai dacă nu are indecşi globali sau constrângeri de integritate referențiale asociate. Fuzionarea unei partiții a unei tabele se face totdeauna cu partiția imediat superioară. Presupunem că avem tabela TAB10 cu partițiileP1, P2, P3 şi P4 şi vrem

să reunim partiția P2 cu P3, se vor exporta datele din tabele P2, se execută comanda **ALTER TALE tab10 DROP PARTITION p2 ,** după care importăm datele exportate în partiția P3.

- **Fuzionarea sau reunirea partițiilor unui index** local se face implicit când se reunesc partițiile corespunzătoare ale acestora, iar fuzionarea a două partiții P2 şi P3, care conțin date, ale unui index global, se poate face astfel:

  ALTER INDEX index_global DROP PARTITION p2;
  ALTER INDEX index_global REBUILD PARTITION p3;

  Schimbarea partițiilor realizează transformarea unei partiții a unei tabele într-o tabelă nepartiționată sau o tabelă nepartiționată într-o partiție a unei tabele partiționate. Operația se execută cu comanda **ALTER TABLE** cu opțiunea **EXCHANGE PARTITION.**

4.6. CREAREA ŞI ACTUALIZAREA VEDERILOR

O ***viziune (vedere*** sau ***tabelă virtuală)*** este o formă de prezentare a datelor din una sau mai multe tabele sau viziuni pentru un utilizator, obținută prin executarea unei cereri. O viziune este tratată ca o tabelă şi se mai numeşte şi *tabelă virtuală*.

Utilizatorul care creează viziunea trebuie să aibă privilegiul CREATE VIEW sau CREATE ANY VIEW. Proprietarul viziunii trebuie să aibă, în mod explicit, acordate privilegiile de acces la toate obiectele referite de către viziune, privilegii ce nu pot fi obținute prin intermediul rolului. De asemenea, funcționalitatea unei viziuni depinde de privilegiile proprietarului acesteia. De exemplu, dacă proprietarul viziunii are privilegiul SELECT pentru tabela din care s-a creat viziunea (numită şi tabelă de bază), atunci acesta poate executa prin intermediul viziunii doar operații de SELECT din tabelă.

Operația se execută cu comanda SQL CREATE VIEW, astfel:

1) ***CREATE VIEW v10 AS***
SELECT col1, col2, col4
FROM tab2
WHERE col1 = 20;

Crearea unei viziuni *v10*, cu coloanele *col1, col2 şi col4*, ca un subset de date din tabela *tab2*, care are coloanele *col1,col2,col3,col4,col5*

2) ***CREATE VIEW v11 AS***
SELECT col1, col2, col4, col7
FROM tab2, tab3
WHERE tab2.col1 = tab3.col1;

Crearea viziunii *v11*ca reuniune a unor date din tabelele tab2 (*col1,col2,col3,col4,col5)* şi tab3 (col1, col8, col9):

Dacă în timpul creării unei viziuni Oracle detectează anumite erori acestea sunt semnalate, iar dacă se foloseşte opţiunea FORCE viziunea este totuşi creată cu starea INVALID. Cu această opţiune o vedere poate fi creată chiar dacă tabela sau tabelele de bază nu există. Viziunea astfel creată va fi validă, deci va pute fi utilizată, abia după ce se va crea tabela de bază, iar proprietarul viziunii va primi drepturile necesare de utilizare a acestora.

Vederea de tip reuniune (vedere join) este definită ca vederea care cumulează rânduri din mai multe tabele. Modificarea unei astfel de vederi se face respectându-se condiţia de *cheie rezervată.* O tabelă se numeşte *tabelă cu cheie rezervată* dacă orice cheie a acesteia poate fi cheie în vederea tip reuniune a cărei tabelă de bază este. Altfel spus, o tabelă cu cheie rezervată are cheile rezervate în cadrul vederii join. Prin intermediul unei astfel de vederi ***se pot actualiza*** date (UPDATE, DELETE sau INSERT) *numai* în tabela de baza care conţine cheia sau cheile rezervate, cu condiţia *obligatorie* ca opţiunea SELECT de creare a vederii să *nu conţină* una din *clauzele* DISTINCT, GROUP BY, START WITH, CONNECT BY, ROWNUM *şi nici o operaţie* de setare de tip UNION, UNION ALL, INTERSECT sau MINUS. Deci prin intermediul unei vederi de tip reuniune se pot modifica date numai asupra coloanelor care se mapează pe tabela de bază care conţine cheia sau cheile rezervate. Cu ajutorul vederilor ALL_UPDATABLE_COLUMNS, DBA_UPDATABLE_COLUMNS şi USER_UPDATABLE_COLUMNS din dicţionarul de date se pot obţine informaţii despre coloanele vederii de tip reuniune ce pot fi modificate.

Inlocuirea unei vederi este operaţia de recrearea acesteia şi se execută prin distrugerea vederii şi recrearea acesteia si redefinirea vederii cu clauza OR REPLACE.

Exemplu:

CREATE OR REPLACE VIEW v10 AS
SELECT col1, col2 FROM tab2
WHERE col1 = 30;

Înainte de a înlocui o vedere, trebuie avute în vedere următoarele efecte:

- Înlocuirea unei vederi determină înlocuirea definiţiei acesteia din dicţionarul de date;
- Dacă în vedere înlocuită a existat clauza CHECK OPTION, iar în definiţia noii vederi nu mai este inclusă, această clauză este distrusă;
- Toate vederile şi programele PL/SQL dependente de vedere înlocuită devin invalide.

Distrugerea vederilor se execută cu comanda **DROP VIEW** .
Vederea partiționată *împarte* o tabelă foarte mare în bucăți mai mici numite partiții şi *le reuneşte* pe acestea pentru a se obține performanțe în administrare şi regăsirea datelor. Cererile care folosesc anumite intervale de valori conforme cu cele folosite la crearea partițiilor vor regăsi date numai din partițiile aferente acestora. O vedere partiționată *se creează* folosind constrângerea de integritate CHECK sau clauza WHERE. Considerăm tabelele t1, t2 şi t3 cu aceleaşi coloane, deci pot fi considerate partiții ale unei tabele care la însumează.

Exemplu:

```
ALTER TABLE t1 ADD CONSTRAINT c1 CHECK (col1 between
0 and 1000);
ALTER TABLE t2 ADD CONSTRAINT c1 CHECK (col1 between
1000 and 10000);
ALTER TABLE t3 ADD CONSTRAINT c1 CHECK (col1 between
10000 and 100000);
CREATE VIEW v10 AS
            SELECT * FROM t1 UNION ALL
            SELECT * FROM t2 UNION ALL
SELECT * FROM t3;

CREATE VIEW v10 AS
            SELECT * FROM t1 WHERE col1 between 0
      and 1000      UNION ALL
            SELECT * FROM t2 WHERE col1 between 1000
      and 10000     UNION ALL
SELECT * FROM t3 WHERE col1 between 10000 and 100000);
```

4.7. CREAREA ŞI ACTUALIZAREA SECVENȚELOR

Secvențele sunt numere unice de identificare a coloanelor unei tabele şi pot fi utilizate la efectuarea diferitelor operații într-o aplicație.

Crearea unei secvențe se execută cu comanda **CREATE SEQUENCE***,* astfel:

```
CREATE SEQUENCE secv_1 INCREMENT BY 1
START WITH 1
NOMAXVALUE
CACHE 10;
```

unde:

INCREMENT BY arată valoare cu care se incrementează o secvență curentă pentru a se obține secvența următoare,

START WITH este valoarea de pornire a secvenței(prima valoare),
NOMAXVALUE arată că nu avem o limită superioară până unde se pot genera secvențe,
CACHE defineşte numărul de secvențe viitoare care se păstrează anticipat în memorie pentru obținerea unor performanțe superioare. Când această valoare se epuizează Oracle încarcă în memorie următorul se de secvențe.

Modificarea unei secvențe se face cu comanda SQL **ALTER SEQUENCE** şi poate opera asupra parametrilor inițiali ai comenzii de creare a secvenței.

Parametrul de inițializare SEQUENCE_CACHE_ENTRIES determină numărul secvențelor care pot fi ținute în memorie de Oracle.

Distrugerea secvențelor se face cu comanda SQL DROP SEQUENCE.

Referirea secvențelor se face cu pseudocoloanele NEXTVAL şi CURRVAL , în care:
NEXTVAL generează următoarea valoare a secvenței. Referirea se face prin *nume_secvență.NEXTVAL.*

Exemple:

1) ***CREATE SEQUENCE secv1;***
INSERT INTO tab1 (COL1, COL2) VALUES (secv1.NEXTVAL, 300);
CURRVAL defineşte valoarea curentă a secvenței şi se referă prin *nume_*

2) ***INSERT INTO tab10 (COL4, COL5) VALUES (secv1.CURRVAL, 1300);***
INSERT INTO tab10 (COL4, COL5) VALUES (secv1.CURRVAL, 2 300);

Valoarea NEXTVAL poate fi referită o singură dată, iar valoarea CURRVAL de mai multe ori, cu condiția ca valoarea NEXTVAL să fi fost referită, deci secvența curentă să fi fost creată.

4.8. CREAREA ŞI ACTUALIZAREA SINONIMELOR

Sinonimul este un alt nume (alias) pentru o tabelă, o vedere, secvență, procedură, funcție sau pachet. Sinonimul poate fi *public* sau *privat.* Sinonimul public este inclus în schema unui grup de utilizatori numit PUBLIC şi este accesibil tuturor utilizatorilor, iar cel privat aparține numai unui anumit utilizator.

Crearea sinonimului se face cu comanda **CREATE SYNONYM** şi se distruge cu comanda **DROP SYNONYM**.

Exemple:

Crearea unui sinonim privat:
CREATE SYNONYM sin1 FOR tab10;
Crearea unui sinonim public:
CREATE PUBLIC SYNONYM sin10 FOR tab10;
Distrugerea unui sinonim:
DROP SYNONYM sin1;
DROP PUBLIC SYNONYM sin10;

4.9. CREAREA ŞI ACTUALIZAREA GRUPURILOR DE TABELE ŞI A GRUPURILOR DE INDECŞI

Grupul de tabele (cluster) este o metodă de memorare comprimată a unor tabele de date care au coloane comune şi care sunt foarte des folosite împreună.

Cheia grupului (key cluster) este coloana sau grupul de coloane pe care tabelele grupate le au comune. Cheia grupului se va specifica atunci când se creează acesta şi atunci când se creează tabelele ce vor fi incluse în grup. Fiecare valoare a cheii de grup se va memora o singură în cadrul grupului de tabele sau al grupului de indecşi indiferent de câte ori apare aceasta în tabelele grupului.

Coloanele care se aleg pentru a fi definite cheile de grup sunt cele folosite cel mai mult pentru a reuni tabelele atunci când se execută o anumită cerere. Cea mai bună cheie de grup este aceea care are suficiente valori unice, astfel încât rândurile care se grupează după aceasta să poată fi incluse într-un singur bloc de date. Astfel dacă avem *prea puține* rânduri pe o cheie de grup obținem o pierdere a spațiului de memorie şi performanțe neglijabile, iar dacă avem *prea multe* timpul de căutare suplimentar, căutare în mai multe blocuri de date, va duce la degradarea performanțelor.

Setarea parametrilor de memorie PCTFREE şi PCTUSED trebuie făcută cu mare grijă, astfel încât să nu afectăm spațiul utilizat pentru inserarea rândurilor şi nici pe cel ce va fi folosi pentru actualizarea datelor aferente rândurilor inserate în bloc. Valorile acestor parametrii folosite la definirea grupului sunt automat utilizate şi pentru tabelele ce vor fi grupate. Chiar dacă vom specifica aceşti parametrii la crearea unei tabele în cadrul grupului ei vor fi ignorați.

Specificarea spațiului necesar pentru memorarea rândurilor aferente unei chei de grup se face prin intermediul clauzei *SIZE* a comenzii SQL *CREATE CLUSTER*. Valoarea acestui parametru este specificată în bytes şi reprezintă spațiul ce trebuie rezervat în cadrul blocului de date aferent grupului pentru memorarea valorii sau valorilor cheii de grup. Prin intermediul acestuia, Oracle determină numărul rândurilor de date ce încap

într-un bloc de date al grupului. Dacă SIZE este specificat astfel încât într-un bloc de date să încapă două chei de grup atunci spațiul acestui bloc este folosit de ambele chei, iar dacă un bloc de date nu poate cuprinde toate rândurile aferente unei chei de grup, cheia de grup se memorează o singură dată, iar blocurile de date se înlănțuiesc de blocul în care se află cheia de grup. Dacă într-un bloc de date încap mai multe chei de grup, atunci acesta poate să aparțină mai multor lanțuri de date.

Specificarea locului (tabelei spațiu) în care să fie plasat grupul de tabele şi grupul index asociat este obligatorie la crearea acestor grupuri.

Crearea grupului de tabele (cluster) se face cu comand SQL *CREATE CLUSTER*. Se va crea întâi grupul de tabele şi apoi tabelele ce vor face parte din grup. Atributele fizice se furnizează o singură dată, doar pentru grup nu şi pentru tabele.

Exemplu:

CREATE CLUSTER grup1 (col1 NUMBER (5))
PCTUSED 75 PCTFREE 10
SIZE 600
TABLESPACE tabsp1
STORAGE (INITIAL 200k
NEXT 290k
MINEXTENTS 3
MAXEXTENTS 25
PCTINCREASE 30);

CREATE TABLE tab1 (col1 NUMBER(5) PRIMARY KEY, col2 NUMBER (10), …) CLUSTER grup1 (col1);
CREATE TABLE tab2 (col3 NUMBER(5) PRIMARY KEY, col4 NUMBER (10), …,
col1 NUMBER (5) REFERENCE tab1(col1)) CLUSTER grup1 (col1);

unde ***col1*** este cheia grupului.

Crearea grupului de indecşi (cluster) se face cu comand SQL *CREATE INDEX cu clauza ON CLUSTER.* Se va crea întâi grupul de tabele şi apoi grupul de indecşi asociat.

Exemplu:

CREATE INDEX index1
ON CLUSTER grup1
INITRANS 2
MAXTRANS 5
TABLESPACE tabsp2GR

PCTFREE 10
STORAGE (INITIAL 50k
NEXT 50k
MINEXTENTS 3
MAXEXTENTS 25
PCTINCREASE 30);

Modificarea grupurilor de tabele sau de indecşi se face cu comanda SQL *ALTER CLUSTER*. Elementele ce pot face obiectul modificării sunt atributele fizice ale grupului.

CAPITOLUL 5. INCARCAREA SI ACTUALIZAREA DATELOR CU COMENZI SQL

5.1. ADĂUGAREA DE NOI TUPLURI

Actualizarea datelor se referă la adăugarea unor noi rânduri într-o tabelă (cu comanda **INSERT**), la modificarea valorilor uneia sau mai multor valori dintr-un rând (cu comanda **UPDATE**) şi la ştergerea unui rând dintr-o tabelă (cu comanda **DELETE**).

În vederea adăugarii unor rânduri noi într-o tabelă sau într-o viziune se utilizează comanda:

INSERT INTO nume-tabelă
[(nume-col1,nume-col2,...)]
{VALUES (valoare 1,valoare2,...) | cerere);

Pentru *nume-col1,nume-col2...* precizate în paranteze vor fi furnizate valorile corespunzătoare, iar coloanelor nespecificate le sunt ataşate valori nule. Coloanele pot fi precizate în orice ordine, însă trebuie asigurată corespondenţa între numele coloanelor şi valorile furnizate.

În cazul în care anumite coloane nu sunt specificate explicit se impune ca ordinea în care apar valorile în comanda **INSERT** să coincidă cu cea în care coloanele au fost definite la crearea tabelei.

Dacă nu se mai cunoaşte ordinea de declarare a coloanelor se foloseşte comanda **DESCRIBE** care va afişa lista coloanelor definite pentru tabela respectivă, tipul şi lungimea lor.

Prin forma **INSERT**...**VALUES** se introduce în tabelă un singur rând. Cu ajutorul valorii **NULL** se pot introduce valori nule. Pentru a furniza valori pentru o coloană de tip dată calendaristică se poate folosi funcţia **TO_DATE** sau cuvântul cheie **SYSDATE**.

Funcţia **TO_DATE** permite furnizarea valorilor într-un format diferit de cel standard.

Specificarea cererii din comanda INSERT determină copierea unor date dintr-o tabelă în alta pe atâtea rânduri câte au rezultat din cererea SQL.

***Exemple*:**

1) Să se adauge un nou rând în tabela PRODUSE.

SQL> INSERT INTO PRODUSE VALUES

1 (100000,11111,'MESE 15/20',7,'27-JUN-92,'BUC')
1 record created.

2) Să se creeze o nouă tabelă, EXEMPLU, cu un singur câmp numeric, identic cu coloana CODP a tabelei PRODUSE. Să se introducă în această nouă tabelă codul produselor din tabela PRODUSE.

SQL> CREATE TABLE EXEMPLU
2 (CODP NUMBER(6) NOT NULL);
Table created.
SQL> INSERT INTO EXEMPLU
2 SELECT CODP FROM PRODUSE;
5 records created.

5.2. MODIFICAREA TUPLURILOR DIN TABELE

În funcție de momentul în care se doreşte realizarea modificărilor asupra bazei de date, utilizatorul poate folosi una din următoarele comenzi: **SET AUTOCOMMIT IMM[EDIATE]** (schimbările se efectuează imediat); **SET AUTOCOMMIT OFF** (schimbările sunt păstrate într-un buffer).La execuția comenzii **COMMIT** se permanentizează schimbările efectuate, iar la execuția comenzii **ROLLBACK** se renunță la schimbările realizate.

În scopul modificării datelor dintr-o tabelă se utilizează una din formele sintactice ale comenzii **UPDATE**:

UPDATE nume-tabelă [sinonim]
SET nume-crt=expresie,nume -expresie,...
[WHERE condiție];

UPDATE nume-tabelă [sinonim]
SET (nume-col, nume-col,...)=(subcerere)
[WHERE condiție];

Sunt două posibilitati de modificare. Una constă în furnizarea în mod explicit a fiecarei valori pentru câmpurile care trebuie modificate iar cealaltă posibilitate constă în obținerea valorilor în urma unei cereri SQL.

Dacă nu este specificată clauza **WHERE** se vor modifica toate rândurile tabelei.

(nume-col,nume-col,...)=(subeercre) impune ca cererea să conțină pentru fiecare rând un număr de valori corespunzător numărului de coloane din

paranteza care precede caracterul =. În cazul în care se modifseă o singură coloană, parantezele pot fi omise.

Exemple:

1) Să se modifice câmpul NRSAL din tabela SALARIAȚI, pentru depozitul cu codul 130000, atribuindu-i valoarea 11.

SQL> UPDATE DEPOZITE
2 SET NRSAL=11
3 WHERE CODD= 130000;
1 record updated.

2) Să se modifice data de livrare, cantitatea solicitată şi prețul de livrare pentru produsul cu codul 13333 din comanda cu numărul 211111.

SQL> UPDATE COMENZI
2 SET DATAL=SYSDATE,CANT=50, PRET=42000
3 WHERE CODP=13333 AND
4 NRCOM=211111;
1 record updated.

3) Să se modifice data de livrare cu data actuală pentru toate produsele cu codul egal cu 13333, din toate comenzile.

SQL> UPDATE COMENZI
2 SET DATAL=SYSDATE
3 WHERE CODP=13333;
1 record updated.

4) Să se mărească salariul cu 15% pentru salariații care au o funcție identică cu CARMEN ANA.

SQL> UPDATE SALARIAȚI
2 SET SALA=SALA*1.15
3 WHERE FUNCT IN
4 (SELECT FUNCT
5 FROM SALARIAȚI
6 WHERE NUME='CARMEN ANA');
11 records updated.

5.3. ŞTERGEREA TUPLURILOR DIN TABELE

Ştergerea unor rânduri dintr-o tabelă se realizează cu următoarea comandă:

DELETE FROM nume tabela [WHERE condiție];

Folosirea clauzei **WHERE** determină ştergerea acelor rânduri care îndeplinesc condiția impusă. În această clauză pot fi folosite şi subcereri. Dacă nu este specificată nici o condiție, se şterg toate rândurile tabelei. Ştergerile accidentale pot fi omise, restaurându-se valorile inițiale prin comanda **AUTOCOMMIT OFF.**

Exemple:

1) Să se şteargă datele din tabela DEPOZITE.

SQL> DELETE FROM DEPOZITE;
5 records deleted.

2) Să se şteargă datele pentru depozitele care au codul mai mare sau egal cu 100000.

SQL> DELETE FROM DEPOZITE
2 WHERE CODD>=100000;
2 records deleted.

3) Să se scrie comanda pentru ştergerea datelor despre salariatul VLAD VASILE. Ştergerile să nu fie efectuate imediat ci ulterior.

SQL> SET AUTOCOMMIT OFF
SQL> DELETE FROM SALARIAȚI
2 WHERE NUME='VLAD VASILE';
1 record deleted.
SQL> COMMIT ;

4) Să se şteargă datele salariaților care au aceeaşi funcție cu a lui PAUL ŞTEFAN. Ştergerile să nu fie realizate imediat. Ulterior să se renunțe la aceste ştergeri.

SQL> SET AUTOCOMMIT OFF
SQL> DELETE FROM SALARIAȚI
2 WHERE FUNCT IN
3 (SELECT FUNCT FROM SALARIAȚI
4 WHERE NUME='PAUL ŞTEFAN');
5 records deleted.

CAPITOLUL 6. SELECTAREA DATELOR DIN TABELELE BAZEI DE DATE

6.1. Comanda SELECT

Pentru a selecta datele din una sau mai multe tabele se utilizează comanda SELECT a carei sintaxa este:

SELECT [ALL | DISTINCT]
{[nume-tabela.]* |
expr [sinonim], expr [sinonim],…}
FROM nume-tabelă [@ legatură][sinonim],…
nume-tabelă [@ legatură][sinonim],
[WHERE conditie]
[CONNECT BY conditie [START WITH conditie]]
[GROUP BY { expr, expr.. | CUBE (expr, expr..) | ROLLUP (expr, expr) }] [HAVING conditie]
[{UNION | INTERSECT | MINUS} SELECT…]
[ORDER BY {expr | număr-poziție} [ASC | DESC]
{expr | numar-pozitie}[ASC | DESC],…
[FOR UPDATE OF nume-col, nume-col,…[NOWAIT]];

Clauzele comenzii trebuie utilizate în ordinea specificată în sintaxă, excepție făcând clauzele **CONNECT BY**, **START WITH**, **GROUP BY** şi **HAVING** (care pot fi specificate în orice ordine). Clauzele **ORDER BY** şi **FOR UPDATE OF** pot fi schimbate între ele. Clauza **ALL** determină afişarea tuturor rândurilor rezultate în urma cererii, spre deosebire de clauza **DISTINCT** care determină eliminarea duplicatelor, afişînd doar rândurile distincte. Utilizarea caracterului asterisc (*) are ca efect selectarea tuturor coloanelor din tabela specificată prin clauza **FROM**, în ordinea în care au fost definite la creare.

În situația finală, fiecare expresie formulată în comandă devine un nume de coloană. Totodată, orice *sinonim*, dacă este specificat, este folosit în scopul etichetării expresiei precedente din tabela afişată. Dacă *sinonimul* conține blancuri sau caractere speciale cum sunt “+” şi ”-“, trebuie incluse între apostrofuri.

Pentru a evita ambiguitatea, în cazul în care tabelele conțin coloane cu acelaşi nume, este necesară calificarea coloanelor cu numele tabelei .

Identificarea tabelelor în care trebuie căutate datele corespunzătoare coloanelor specificați se face prin specificarea numelor lor după clauza **FROM**. În locul numelor de tabele se pot folosi sinonimele acestora.

Clauza **WHERE** specifică o condiție care este folosită pentru a selecta rândurile.

Clauza **CONNECT BY** indică faptul că rândurile formează o structură arborescentă. Prin această clauză sunt definite relațiile necesare pentru a conecta rândurile tabelei într-un arbore. Operatorul **PRIOR** folosit înaintea uneia din cele două părți ale condiției, defineşte nodul părinte iar în cealaltă parte nodul fiu. Clauza **START WITH**, prin specificarea unei condiții care trebuie satisfăcută, stabileşte rândul folosit ca rădăcină a arborelui. Dacă se omite, comanda **SELECT** va returna o serie de arbori începând cu fiecare rând selectat. Existența clauzei **CONNECT BY** într-o comandă **SELECT** permite utilizarea pseudocoloanei **LEVEL**, care returnează valoarea 1 pentru nodul rădăcină, 2 pentru fiii nodului rădăcină, 3 pentru nepoți etc.

Clauzele **GROUP BY** şi **HAVING** determină afişarea unor informații sintetice despre grupuri de rânduri care au aceeaşi valoare în una sau mai multe coloane. Aceste coloane sunt, în general, funcții de grup.

ROLLUP activează o comanda SELECT pentru a calcula mai multe niveluri de subtotaluri dintr-un grup specificat de dimensiuni. Calculează de asemenea şi un total general. ROLLUP este o extensie simplă a clauzei GROUP BY, deci sintaxa este foarte uşor de folosit. Extensia ROLLUP este foarte eficientă şi nu îngreunează o cerere. Rolul extensiei ROLLUP este foare clar: crează subtotaluri care pornesc de la cel mai detaliat nivel până la un total general după gruparea care a fost precizată în clauza ROLLUP.

CUBE acționează asupra unui grup specificat de coloane şi crează subtotaluri pentru toate combinațiile posibile între acestea. Dacă s-a specificat de exemplu: CUBE (timp,regiune, department), rezultatul va include toate valorile care ar fi incluse într-un ROLLUP plus combinații adiționale.

Pentru a combina rezultatele a două comenzi SELECT într-un singur rezultat, se folosesc operatorii UNION, INTERSECT şi MINUS. **UNION** returnează rezultatele obținute de la fiecare cerere în parte, **INTERSECT** returnează doar rezultatele comune celor două cereri iar **MINUS** returnează rezultatele obținute de la prima cerere şi care nu apar în urma celei de a doua selecții.

Pentru a putea folosi aceste clauze este necesar ca numărul şi tipul coloanelor selectate de fiecare comandă **SELECT** să fie aceleaşi, lungimile lor putând fi diferite. Dacă sunt combinate mai mult de două comenzi **SELECT**, ele vor fi evaluate de la stînga la dreapta. Pentru a schimba ordinea de evaluare pot fi folosite parantezele. Totodată, cei trei operatori impun utilizarea cuvântului **DISTINCT** în toate comenzile **SELECT**.

Clauza **ORDER BY** specifică ordinea în care trebuie returnate rândurile distincte ale unei tabele.

Clauza **FOR UPDATE** determină blocarea rândurilor selectate ale tabelei astfel încât acestea nu vor mai putea fi actualizate de alți utilizatori până la deblocarea lor cu una din comenzile **COMMIT** sau **ROLL BACK.**

Comanda **SELECT ... FOR UPDATE** trebuie urmată de una sau mai multe comenzi **UPDATE ... WHERE**. Dacă se foloseşte clauza **NOWAIT**, selecția este considerată terminată chiar dacă rândurile selectate de **FOR UPDATE** nu pot fi blocate deoarece alt utilizator lucrează cu ele.

6.2. Utilizarea clauzei FROM

Pentru a selecta una sau mai multe tabele şi pentru a specifica, eventual, identificatorul proprietarului şi legătura cu rețeaua se foloseşte secvența:

SELECT ...
FROM [ident-proprietar] tabela [@LINK],...;

Exemple:

1) Să se selecteze toate coloanele din tabela SALARIAȚI.
SQL> SELECT * FROM SALARIAȚI; sau
SQL> SELECT ALL FROM SALARIATI;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
1111	AVRAM ION	VÂNZATOR	100000	21200	1000	1000
1222	BARBU DAN	VÂNZATOR	120000	20750	2000	1000
1000	COMAN RADU	ŞEF DEP	130000	35000	2500	1000
3500	DAN ION	VÂNZATOR	160000	24500	3550	2500
2500	VLAD VASILE	ŞEF DEP	160000	36500	1500	2500
3700	MANU DAN	VÂNZATOR	160000	27500	2500	2500
2650	VLAD ION	VÂNZATOR	120000	25060	3500	1000

7 records selected.

2) Să se selecteze coloanele MARCA, NUME, SALA, VENS din tabela SALARIAȚI.

SQL> SELECT MARCA,NUME,SALA,VENS
2 FROM SALARIAȚI;

MARCA	NUME	SALA	VENS
1111	AVRAM ION	21200	1000
1222	BARBU DAN	20750	2000
1000	COMAN RADU	35000	2500
3500	DAN ION	24500	3550
2500	VLAD VASILE	36500	1500
3700	MANU DAN	27500	2500
2650	VLAD ION	25060	3500

7 records selected.

3) Să se selecteze coloana NUME din tabela SALARIAȚI

SQL> SELECT NUME
2 FROM SALARIAȚI;

NUME
AVRAM ION
BARBU DAN
COMAN RADU
DAN ION
VLAD VASILE
MANU DAN
VLAD ION

7 records selected.

4) Să se selecteze coloana FUNCT din tabela SALARIAȚI în variantele utilizării şi neutilizării clauzei DISTINCT.

SQL> SELECT FUNCT AFIŞARE_FUNCTIE
2 FROM SALARIAȚI;

AFISARE_FUNCTIE
VÂNZATOR
VÂNZATOR
SEF DEP
VÂNZATOR
SEF DEP
VÂNZATOR
VÂNZATOR

7 records selected.

SQL>SELECT DISTINCT FUNCT
2 AFIŞARE_DISTINCT_FUNCTIE
3 FROM SALARIAȚI;

AFISARE_DISTINCT_FUNCTIE
VÂNZATOR
ŞEF DEP
2 records selected.

6.3 Utilizarea operatorilor în formularea condițiilor de selecție din clauza WHERE

Operatorii SQL*Plus (Anexa2) pot apărea în orice parte a unei comenzi şi au întâietate faţă de orice alt tip de operatori. Operatorii aritmetici şi logici sunt utilizaţi pentru formularea condiţiilor de selecţie.

Exemple:
1) Să se selecteze toate înregistrările privind salariaţii al căror salariu este mai mare de 15000 u.m.:

SQL> SELECT * FROM SALARIATI
2 WHERE SALA>15000 ;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
1111	AVRAM ION	VÂNZATOR	100000	21200	1000	1000
1222	BARBU DAN	VÂNZATOR	120000	20750	2000	1000
1000	COMAN RADU	ŞEF DEP	130000	35000	2500	1000
3500	DAN ION	VÂNZATOR	160000	24500	3550	2500
2500	VLAD VASILE	ŞEF DEP	160000	36500	1500	2500
3700	MANU DAN	VÂNZATOR	160000	27500	2500	2500
2650	VLAD ION	VÂNZATOR	120000	25060	3500	1000

7 records selected.

2) Să se selecteze coloanele MARCA şi NUME, precum şi veniturile totale (SALA+VENS) pentru angajaţii care au un salariu mai mare decât 35.000 u.m.

SQL> SELECT MARCA, NUME, SALA+VENS
2 FROM SALARIAȚI
3 WHERE SALA>35000 ;

MARCA	NUME	SALA + VENS
2650	VLAD VASILE	38000

1 record selected

3) Să se selecteze coloanele NUME, FUNCT şi SALA pentru salariaţii care au fncţia de vânzător.

SQL> SELECT NUME, FUNCT FROM SALARIAȚI
2 WHERE FUNCT='VÂNZATOR' ;

NUME	FUNCT	SALA
AVRAM ION	VÂNZATOR	21200

BARBU DAN	**VÂNZATOR**	**20750**
DAN ION	**VÂNZATOR**	**24500**
MANU DAN	**VÂNZATOR**	**27500**
VLAD ION	**VÂNZATOR**	**25060**

5 records selected.

4) Să se selecteze coloanele MARCA, NUME, CODD din tabela SALARIAȚI, pentru salariații al căror venit suplimentar depăşeşte salariul.

SQL> SELECT MARCA, NUME, CODD
2 FROM SALARIAȚI
3 WHERE VENS>SALA ;
no records selected

5) Să se selecteaze şi afişeze câmpurile MARCA, NUME, SALARIU pentru salariații ale căror venituri suplimentare sunt mai mari de 1.500 u.m. şi lucrează în subordinea superiorului cu Codul 1000.

SQL> SELECT MARCA, NUME, SALA
2 FROM SALARIAȚI
3 WHERE VENS>1500
4 AND CODS=1000 ;

MARCA	NUME	SALA
1222	BARBU DAN	20750
1000	COMAN RADU	35000
2650	VLAD ION	25060

3 records selected

6) Să se afişeze toate coloanele pentru salariații cu funcția vânzător, care au salariul mai mare ca 20.000 u.m. şi lucrează în subordinea superiorului cu Codul 1000.

SQL> SELECT * FROM SALARIAȚI
2 WHERE FUNCT ='VANZATOR'
3 AND SALA>20000
4 AND CODS=1000 ;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
1111	AVRAM ION	VÂNZATOR	100000	21200	1000	1000
1222	BARBU DAN	VÂNZATOR	120000	20750	2000	1000
2650	VLAD ION	VÂNZATOR	120000	25060	3500	1000

3 records selected.

7) Să se afişeze coloana NUME şi SALA pentru angajații care au salariul mai mic ca 30.000 u.m.

SQL> SELECT NUME FROM SALARIAȚI

2 WHERE SALA<30000 ;

NUME	SALA
AVRAM ION	21200
BARBU DAN	20750
DAN ION	24500
MANU DAN	27500
VLAD ION	25060

5 records selected.

8) Să se selecteze înregistrările pentru care funcţia este 'SEF DEP' sau salariul este mai mare decât 35.000 u.m.

SQL> SELECT MARCA, CODS FROM SALARIAŢI
2 WHERE FUNCT='SEF DEP' OR SALA>35000 ;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
1000	COMAN RADU	ŞEF DEP	130000	35000	2500	1000
2500	VLAD VASILE	ŞEF DEP	160000	36500	1500	2500

2 records selected.

9) Să se selecteze datele despre salariaţii care au funcţia de vânzător şi nu lucrează în subordinea superiorului cu codul 1000,

SQL> SELECT * FROM SALARIAŢI
2 WHERE FUNCT='VANZATOR'
3 AND CODS!=l000;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
3500	DAN ION	VÂNZATOR	160000	24500	3550	2500
3700	MANU DAN	VÂNZATOR	160000	27500	2500	2500

2 records selected.

10) Să se selecteze toţi salariaţii care lucrează în subordinea superiorului cu codul 1000 precum şi cei care au salariul mai mic de 26.000 u.m. sau funcţia de vânzător.

SQL> SELECT * FROM SALARIAŢI
2 WHERE FUNCT='VÂNZATOR'
3 OR SALA< 26000 AND CODS=1000 ;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
1111	AVRAM ION	VÂNZATOR	100000	21200	1000	1000
1222	BARBU DAN	VÂNZATOR	120000	20750	2000	1000
3500	DAN ION	VÂNZATOR	160000	24500	3550	2500
2650	VLAD ION	VÂNZATOR	120000	25060	3500	1000

4 records selected

11) Să se selecteze MARCA şi NUMELE pentru înregistrările care conţin date despre angajaţii a căror funcţie este cea de şef de depozit sau

care au un salariu de 35.000 u.m. şi lucrează în subordinea superiorului cu codul 1000.

SQL> SELECT MARCA, NUME
2 FROM SALARIAȚI
3 WHERE FUNCT=’SEF DEP’ OR
4 (SALA=35000 AND CODS=1000) ;

MARCA	NUME
1000	COMAN RADU
2500	VLAD VASILE

2 records selected.

12) Să se selecteze datele salariaților care lucrează în subordinea superiorului cu marca 1000 şi au funcția şef de depozit sau salariul în valoare de 35.000 u.m.

SQL> SELECT * FROM SALARIAȚI
2 WHERE (FUNCT=’SEF DEP’ OR SALA=35000)
3 AND CODS=1000 ;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
1000	COMAN RADU	ŞEF DEP	130000	35000	2500	1000

1 record selected.

13) Să se afişeze valorile coloanelor MARCA, NUME, FUNCT privind angajații care lucrează în subordinea superiorului cu marca 1000 şi au funcția de şef de depozit sau de vânzător.

SQL> SELECT MARCA,NUME,FUNCT, CODS
2 FROM SALARIAȚI
3 WHERE (FUNCT=’SEF DEF’ OR
4 FUNCT=’VANZATOR')
5 AND CODS=1000 ;

MARCA	NUME	FUNCT	CODS
1111	AVRAM ION	VÂNZATOR	1000
1222	BARBU DAN	VÂNZATOR	1000
1000	COMAN RADU	ŞEF DEP	1000
2650	VLAD ION	VÂNZATOR	1000

4 records selected.

14) Să se selecteze coloanele MARCA, NUME, FUNCT pentru salariații care au funcția de şef depozit sau pentru cei care lucrează în subordinea superiorului cu marca 1000 şi au funcția de vânzător.

SQL> SELECT MARCA,NUME,FUNCT, CODS
2 FROM SALARIAȚI
3 WHERE FUNCT='SEF DEP'

4 OR (FUNCT='VÂNZATOR'
5 AND CODS=1000) ;

MARCA	NUME	FUNCT	CODS
1111	AVRAM ION	VÂNZATOR	1000
1222	BARBU DAN	VÂNZATOR	1000
1000	COMAN RADU	ŞEF DEP	1000
2500	VLAD VASILE	ŞEF DEP	2500
2650	VLAD ION	VÂNZATOR	1000

5 records selected.

15) Să se selecteze toate datele privind angajații ce nu au funcția de vânzător.

SQL> SELECT * FROM SALARIAȚI
2 WHERE NOT (FUNCT= 'VANZATOR') ;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
1000	COMAN RADU	ŞEF DEP	130000	35000	2500	1000
2500	VLAD VASILE	ŞEF DEP	160000	36500	1500	2500

2 records selected.

16) Să se selecteze valorile coloanelor MARCA, NUME, FUNCT, SALA+VENS pentru angajații care au salariul cuprins între 24.500 şi 36.000 u.m.

SQL>SELECT MARCA, NUME, FUNCT, SALA+VENS
2 FROM SALARIAȚI
3 WHERE SALA BETWEEN 24500 AND 36000 ;

MARCA	NUME	FUNCT	SALA+VENS
1000	COMAN RADU	ŞEF DEP	37500
3500	DAN ION	VÂNZATOR	28050
3700	MANU DAN	VÂNZATOR	30000
2650	VLAD ION	VÂNZATOR	28560

4 records selected.

17) Să se selecteze câmpurile NUME, FUNCT, SALA+VENS pentru salariații care au salariul mai mic decât 24500 şi mai mare decât 36000.

SQL> SELECT NUME, FUNCT, SALA+VENS
2 FROM SALARIAȚI
3 WHERE SALA NOT BETWEEN 24500 AND 36000 ;

NUME	FUNCT	SALA+VENS
AVRAM ION	VÂNZATOR	22200
BARBU DAN	VÂNZATOR	22750
VLAD VASILE	ŞEF DEP	38000

3 records selected.

18) Să se selecteze câmpurile MARCA, NUME şi FUNCT pentru salariaţii care lucrează în depozitul cu codurile 130000 sau 160.000.

SQL> SELECT MARCA, NUME, FUNCT
2 FROM SALARIAŢI
3 WHERE CODD IN (130000,160000) ;

MARCA	NUME	FUNCT
1000	COMAN RADU	ŞEF DEP
3500	DAN ION	VÂNZATOR
2500	VLAD VASILE	ŞEF DEP
3700	MANU DAN	VÂNZATOR

4 records selected

19) Să se selecteze câmpurile MARCA, NUME, SALA, VENS pentru salariaţii care au altă funcţie decât cea de vânzător.

SQL> SELECT MARCA, NUME, SALA, VENS
2 FROM SALARIAŢI
3 WHERE FUNCT NOT IN ('VÂNZATOR') ;

MARCA	NUME	FUNCT	SALA	VENS
1000	COMAN RADU	ŞEF DEP	35000	2500
2500	VLAD VASILE	ŞEF DEP	36500	1500

2 records selected

Operatorul LIKE face posibilă realizarea unor cereri de regăsire a înregistrărilor ce conţin un câmp a cărui valoare este comparată cu un şir de caractere dat. Pentru a evalua expresiile logice în care apar şiruri de caractere se poate folosi clauza **LIKE** în următoarea secvenţă:

SELECT ...
WHERE coloană LIKE şir …;

Avantajul unei viteze mari de regăsire ca urmare a indexării este pierdut în momentul în care se caută într-o coloană indexată un şir care începe cu **"-"** sau **"%"**. Aceste caractere speciale suplinesc unul, respectiv mai multe caractere.

Exemple:

1) Să se selecteze coloanele NUME şi FUNCT precum şi veniturile totale pentru salariaţii al căror nume începe cu litera C.

SQL> SELECT NUME, FUNCT, SALA+VENS
2 FROM SALARIAŢI
3 WHERE NUME LIKE 'C%' ;

NUME	FUNCT	SALA+VENS

COMAN RADU ŞEF DEP 37500

1 record selected

2) Să se selecteze coloanele NUME, FUNCT, SALA+VENS pentru salariații al căror nume se termină cu litera N.

SQL> SELECT NUME,FUNCT, SALA+VENS
2 FROM SALARIAȚI
3 WHERE NUME LIKE '%N' ;

NUME	FUNCT	SALA+VENS
AVRAM ION	VÂNZATOR	22200
BARBU DAN	VÂNZATOR	22750
DAN ION	VÂNZATOR	28050
MANU DAN	VÂNZATOR	30000
VLAD ION	VÂNZATOR	28560

5 records selected

3) Să se selecteze coloanele MARCA, NUME, MARCA, FUNCT pentru salariații al căror nume este format din nouă caractere (inclusiv spațiu), pe ultima poziție fiind N.

SQL> SELECT NUME, MARCA, FUNCT, SALA+VENS
2 FROM SALARIAȚI
3 WHERE NUME LIKE '_ _ _ _ _ _ _ _N' ;

MARCA	NUME	FUNCT
1111	AVRAM ION	VÂNZATOR
1222	BARBU DAN	VÂNZATOR

2 recors selected

4)Să se selecteze salariații al căror nume are pe poziția a treia litera M.

SQL> SELECT NUME, FUNCT
2 FROM SALARIAȚI
3 WHERE NUME LIKE '_ _M%' ;

NUME	FUNCT
COMAN RADU	ŞEF DEP

1 record selected

5) Să se selecteze salariații al căror nume are o lungime de nouă caractere.

SQL> SELECT NUME
2 FROM SALARIAȚI
3 WHERE NUME LIKE '_ _ _ _ _ _ _ _ _';

MARCA	NUME	NS
1111	AVRAM ION	
1222	BARBU DAN	

2 recors selected

Regăsirea unor înregistrări ce conțin câmpuri cu valori nule se face cu ajutorul operatorului NULL.

Exemple:

1) Să se selecteze datele pentru salariații ce nu au venit suplimentar.

SQL> SELECT MARCA, NUME, FUNCT
2 FROM SAI.ARIATI
3 WHERE VENS IS NULL ;

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
3770	CARMEN ANCA	VÂNZATOR	130000	26500	4000	

1 record selected

2) Să se selecteze toate datele salariaților care sunt şef depozit şi al căror câmp VENS (venituri suplimentare) nu conține valoarea NULL.

SQL> SELECT * FROM SALARIATI WHERE
2 VENS IS NOT NULL AND FUNCT IS 'SEF DEP' ;

MARCA	NUME	FUNCT	CODD SALA		VENS	CODS
1000	COMAN RADU	ŞEF DEP	130000 35000		2500	1000
2500	VLAD VASILE	ŞEF DEP	160000 36500		1500	2500

2 records selected

6.4.Ordonarea liniilor rezultate în urma unei cereri

Limbajul SQL*Plus are posibilitatea ordonării crescătoare sau descrescătoare a liniilor rezultate în urma unei cereri. Această operație se realizează prin intermediul secvenței:

SELECT ...
FROM ...
WHERE ...
ORDER BY {expr | număr-poziție} [ASC | DESC], …;

unde:
expr reprezintă o expresie care face referire la una sau mai multe coloane; *număr-poziție* este un număr care identifică *poziția coloanei* din comanda **SELECT**, după care se doreşte *sortarea*. Utilizarea operatorului de tipul **UNION**, **INTERSECT** sau **MINUS** impune prezența argumentului *număr-poziție*. **ASC** sau **DESC** precizează modul

de ordonare a*scendent*, respectiv *descendent*. În cazul în care clauzele **ORDER BY** şi **DISTINC**T sunt utilizate împreună, clauza **ORDER BY** trebuie să se refere la *coloane* care n-au fost menționate în comanda **SELECT**.

Exemple:

1) Să se selecteze toate coloanele tabelei SALARIAȚI privind salariații care sunt vânzători şi pentru care marca superiorului este 1000. Afişarea să se facă după valorile coloanei MARCA, descrescător.

```
SQL>SELECT * FROM SALARIAȚI
2      WHERE CODS=1000
3      AND FUNCT='VÂNZATOR'
4      ORDER BY MARCA DESC ;
```

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
2650	VLAD ION	VÂNZATOR	120000	25060	3500	1000
1222	BARBU DAN	VÂNZATOR	120000	20750	2000	1000
1111	AVRAM ION	VÂNZATOR	100000	21200	1000	1000

3 records selected

2) Să se selecteze crescător după salariu, angajații cu funcția vânzător pentru care marca superiorului este 1000.

```
SQL>SELECT * FROM SALARIAȚI
2      WHERE CODS=1000
3      AND FUNCT='VÂNZATOR'
4      ORDER BY SALA ;
```

MARCA	NUME	FUNCT	CODD	SALA	VENS	CODS
1222	BARBU DAN	VÂNZATOR	120000	20750	2000	1000
1111	AVRAM ION	VÂNZATOR	100000	21200	1000	1000
2650	VLAD ION	VÂNZATOR	120000	25060	3500	1000

3 records selected.

3) Să se selecteze crescător, după salariu, coloanele MARCA, NUME, SALA, VENS, SALA+VENS pentru acei salariați cu funcția de vânzător şi pentru care marca superiorului este 1000

```
SQL> SELECT MARCA, NUME, SALA, VENS
2      FROM SALARIAȚI
3      WHERE CODS=1000 AND FUNCT='VÂNZATOR'
4      ORDER BY SALA ;
```

MARCA	NUME	SALA	VENS	SALA+VENS
1222	BARBU DAN	20750	2000	22750
1111	AVRAM ION	21200	1000	22200
2650	VLAD ION	25060	3500	28560

3 records selected.

4) Să se selecteze coloanele MARCA, NUME, CODD, VENS ordonate crescător după codul depozitului şi veniturile suplimentare.

```
SQL> SELECT MARCA,NUME,CODD,VENS
2        FROM SALARIAȚI
3        WHERE FUNCT='VÂNZATOR'
4        ORDER BY CODD, VENS ;
```

MARCA	NUME	CODD	VENS
1111	AVRAM ION	100000	1000
1222	BARBU DAN	120000	2000
2650	VLAD ION	120000	3500
3700	MANU DAN	160000	2500
3500	DAN ION	160000	3550

5 records selected

6.5. Selecții din mai multe tabele

Operația prin care se selectează şi se grupează coloanele din tabele diferite, în scopul obținerii unor informații coerente, poartă numele de joncțiune (JOIN). Pentru a realiza o *joncțiune* şi pentru a preciza corespondența între rândurile tabelelor, se utilizează următoarea secvență:

SELECT ...
FROM nume-tab1, nume-tab2, ...
WHERE condiție …;

unde:
condiție reprezintă orice expresie care compară câmpurile diferitelor tabele.Vor fi selectate rândurile pentru care condiția este îndeplinită. În condiție poate fi folosit operatorul (+) care simulează existența unor rânduri vide în tabelele pentru care nu se găsesc corespondențe.

Exemple:
1) Din tabela COMENZI (creată anterior de utilizator, pe structura NrCom, DataCom, CodP, Cant, PrețMax, PretMin, Pret(plătit), să se selecteze codul produsului, cantitatea şi diferența dintre valoarea mărfurilor comandate la prețul maxim şi cel efectiv negociat (pentru un câmp definit DIFERENTA). Ordonarea să se facă ascendent după CODP.

```
SQL> SELECT COMENZI.CODP,CANT,
2        PRETMAX*CANT-PRET*CANT DIFERENTA
```

```
3        FROM COMENZI,PRETURI
4        WHERE PRETURI.CODP=COMENZI.CODP
5        ORDER BY COMENZI.CODP ;
```

CODP	CANT	DIFERENTA
13333	4	8000
14444	6	3000
16686	15	75000

3 records selected.

2) Să se afişeze din tabela COMENZI codul produsului, diferența dintre valoarea mărfurilor comandate la prețul maxim şi cel efectiv negociat (pentru DIF_MAX), diferența dintre valoarea mărfurilor comandate la prețul minim şi cel efectiv negociat (pentru DIF_MIN) selectate după criteriul egalității EQUI-JOIN (JOIN pe condiție de egalitate). Liniile vor fi ordonate crescător după valoarea totală a comenzii.

```
SQL> SELECT COMENZI.CODP,
2        PRETMAX*CANT-*PRET*CANT DIF_MAX,
3        PRETMIN*CANT-PRET*CANT DIF_MIN
4        FROM COMENZI.PRETURI
5        WHERE PRETURI.CODP=COMEN2I.CODP
6        ORDER BY PRET*CANT;
```

CODP	DIF_MAX	DIF_MIN
14444	3000	0
13333	8000	-4000
16666	75000	-15000

3 records selected.

3) Să se afişeze câmpurile CODP, DENP, STOC şi CANT, utilizând criteriul egalității OUTER-JOIN, pe câmpul comun CODP (se afişează datele despre acele produse pentru care există comenzi dar nu sunt în tabela Produse). Liniile vor fi ordonate crescător după câmpul CODP.

```
SQL> SELECT COMENZI.CODP, DENP, STOC, CANT
2        FROM PRODUSE, COMENZI
3        WHERE COMENZI.CODP=PRODUSE.CODP (+)
4        ORDER BY COMENZI.CODP ;
```

CODP	DENP	STOC	CANT
13333	CANAPEA A7	6	4
14444	SCAUN D4	36	6
16666	PLACAJ 2/2	100	15

3 records selected.

4) Să se afişeaze în modul distinct (fară a se repeta aceleaşi linii), Codul depozitului şi funcțiile care conțin valori nule în câmpul veniturilor

suplimentare. Selectarea să se facă pe criteriul egalității pe câmpul comun CODD şi în condițiile în care acel depozit există.

SQL> SELECT DISTINCT DEPOZITE.CODD, NUME, FUNCT
2 FROM SALARIATI, DEPOZITE
3 WHERE SALARIATI.CODD=DEPOZITE.CODD (+)
4 AND VENS IS NULL;

CODD	NUME	FUNCT
130000	CARMEN ANCA	VÂNZATOR

1 record selected.

5) Să se afişeze în modul distinct codul depozitului şi funcțiile care conțin valori nule în câmpul veniturilor suplimentare. Selectarea se face pe criteriul egalității în câmpul comun CODD şi în condițiile în care acel depozit există. Liniile vor fi ordonate descrescător după codul depozitului (din tabela SALARIAȚI). Se va adăuga în tabela salariați un nou tuplu cuprinzând datele vânzătorului Alexe Ioan, cod depozit 160000 şi fără venituri suplimentare, pentru eficiența ordonării.

SQL> SELECT DISTINCT DEPOZTTE.CODD.FUNCT
2 FROM SALARIATLDEPOZITE
3 WHERE SALARIATLCODD=DEPOZITE.CODD(+)
4 AND VENS IS NULL
5 ORDER BY SALARIATI.CODD DESC

CODD	NUME	FUNCT
160000	ALEXE IOAN	VÂNZĂTOR
130000	CARMEN ANCA	VÂNZATOR

2 records selected

Într-o cerere se pot înlocui numele de tabele sau coloane prin etichete.

6) Să se selecteze câmpurile NUME, FUNCT, DEND din tabelele DEPOZITE (pentru care este utilizat numele D), CodD, DenD, Capac, NrSal, şi SALARIAȚI (pentru care este utilizat numele S). Liniile vor fi ordonate crescător după câmpul NUME.

SQL> SELECT NUME, FUNCT, D.CODD, DEND
2 FROM DEPOZITE D, SALARIAȚI S
3 WHERE D.CODD=S.CODD
4 ORDER BY NUME

NUME	FUNCT	CODD	DEND
ALEXE IOAN	VÂNZATOR	160000	SPORT
AVRAM ION	VÂNZATOR	100000	MOBILA
BARBU DAN	VÂNZATOR	120000	ALIMENTAR

CARMEN ANCA	VÂNZATOR	130000	AUTO
COMANRADU	SEF DEP	130000	AUTO
DAN ION	VÂNZATOR	160000	SPORT
DORU DAN	SEF DEP	130000	AUTO
FRINCU ION	SEF DEPR	160000	SPORT
RADU IOANA	VÂNZATOR	130000	AUTO
SANDU ION	VÂNZATOR	130000	AUTO
VLAD ION	VÂNZATOR	120000	ALIMENTAR
VLAD VASILE	SEF DEP	160000	SPORT

13 records selected.

Pentru a pune în evidență posibilitatea schimbării denumirilor de tabele, în exemplele care urmează se va folosi, cu preponderență, adresarea prin calificare.

7) Să se selecteze, în condițiile redenumirii tabelelor COMENZI cu CONTRACT şi PRODUSE cu COS, a coloanelor CODP, CANT, PREȚ din CONTRACT şi coloanelor DENP, STOC din COS. Să fie selectate doar câmpurile care au unitatea de măsură "buc”. Liniile vor fi ordonate crescător după valorile comenzilor.

```
SQL> SELECT CONTRACT.CODP,CONTRACT.CANT,
2        CONTRACT.PRET, DENP, COS.STOC
3        FROM COMENZI CONTRACT, PRODUSE COS
4        WHERE CONTRACT.CODP=COS.CODP
5        AND UM='BUC
6        ORDER BY CONTRACT.CANT*COS.PRET
```

CODP	CANT	PREȚ	DENP	STOC
14444	6	4500	SCAUN D4	36
13333	4	80000	CANAPEA A	7 6

2 records selected.

8) Să se selecteze, în condițiile redenumirii tabelelor COMENZI cu CONTRACT şi PRODUSE cu COS, a coloanelor CODP, CANT, PREȚ din CONTRACT şi a coloanelor DENP, STOC din COS. Vor fi selectate doar câmpurile care au unitatea de măsură “buc” iar liniile vor fi ordonate crescător după valorile comenzilor. La afişare se ve extrage o coloană numită 'DIFERENȚA' care să reflecte stocul rămas în urma onorării comenzii.

```
SQL> SELECT CONTRACT.CODP,
2        CONTRACT.CANT,CONTRACT.PRET,
3        DENP, COS.STOC,
4        COS.STOC-CONTRACT.CANT DIFERENȚA
5        FROM COMENZI CONTRACT,PRODUSE COS
6        WHERE CONTRACT.CODP=COS.CODP
7        AND    UM=”BUC'
```

*8 ORDER BY CONTRACT.CANT * CONTRACT.PRET*

CODP	CANT	PREȚ	DENP	STOC	DIFERENȚA
14444	6	4500	SCAUN D4	36	30
13333	4	80000	CANAPEA A7	6	2

2 records selected.

9) Din tabela COMENZI (definită prin etichetele T1 şi T2) să se selecteze CODP, CODC, în condițiile în care valoarea comenzii este mai mare ca 50.000 u.m. (JOIN-ul unei tabele pe ea însăşi).

SQL> SELECT DISTINCT T1.CODP, T2.CODC
2 FROM COMENZI T1,COMENZI T2
3 WHERE Tl.CANT * T2.PRET >60000

CODP	CODC
13333	121111
14444	121111
16666	121111
22222	121111

4 records selected

10) Să se selecteze campurile CODP,.DENP, PREȚ, PRETMAX, PRETM1N pentru produsele al căror preț (negociat) este cuprins între prețul maxim şi prețul minim. Ordonarea să se facă crescător după câmpul CODP.

SQL> SELECT PRODUSE.CODP, DENP,
2 COMENZI.PRET,
3 PRETMIN,PRETMAX
4 FROM PRODUSE, PRETURI, COMENZI
5 WHERE COMENZI.PRET BETWEEN PRETMIN AND PRETMAX
6 ORDER BY PRODUSE.CODP

CODP	DENP	PREȚ	PRETMIN	PRETMAX
13333	CANAPEA A7	80000	79000	82000
14444	SCAUN D4	4500	4500	5000
16666	PLACAJ 2/2	25000	24000	30000

3 records selected.

6.6. Realizarea cererilor incluse

Subcererile reprezintă cereri incluse în clauzele unor comenzi SQL. Rândurile selectate de o subcerere nu sunt afişate, ele fiind utilizate în continuare de o comandă SQL.

Dacă subcererea este folosită în partea dreaptă a unei expresii logice sau a unei expresii de atribuire, ea va returna o singură valoare sau o coloană de valori. Compunerea valorii rezultate cu cea din stânga expresiei se face în conformitate cu operatorul care face legătura între cele două părți.

În cazul în care subcererea este folosită pentru a specifica valori în comenzi ca INSERT, CREATE TABLE, UPDATE, ea va returna câte o valoare pentru fiecare coloană specificată în comandă. Clauze ca ORDER BY, FOR nu pot fi folosite în subcereri.

Subcererile apar, în general, în următoarele comenzi:

COPY [FROM nume-utilizator/parolă@bază-de-date]
[TO nume-utilizator/parolă@bază-de-date]
{APPEND | CREATE | INSERT | REPLACE}
Tabelă (col1,col2,…) USING cerere;…;

CREATE TABLE tabelă …
AS cerere ;

INSERT INTO tabelă | [(col1,col2,…)]
[VALUES (val1,val2,...) | cerere];

UPDATE tabelă [sinonim]
SET (col1, col2, …) = (cerere)
[WHERE condiție];

Există si *subcereri corelate* cu cererile din comanda principală, care apar doar în clauza WHERE a comenzii SELECT. Ele pot utiliza sinonime pentru tabela precizată în comanda SELECT şi sunt evaluate câte o dată pentru fiecare rând selectat în comanda principală. Subcererile corelate pot apare în formule ca:

1). SELECT
coloana1, coloana2,...
FROM tabela1, tabela2 tab2,...
WHERE coloana1 IN
(SELECT coloana1
FROM tabela1
WHERE condiție)
.... ;

2) SELECT
coloana1, coloana2,...
FROM tabela1 tab1, tabela2 tab2,...
WHERE coloana2 IN
(SELECT funcție (coloană)
FROM tabela2 tab2
WHERE tab2.coloana=coloana)
.... ;

Exemple:

1) Să se selecteze câmpurile NUME şi FUNCT ale salariaților cu funcția identică cu a lui RADU IOANA.

```
SQL> SELECT NUME, FUNCT, FUNCȚIE
2        FROM SALARIAȚI
3        WHERE FUNCT=
4        (SELECT FUNCT FROM SALARIAȚI
5        WHERE NUME='RADU IOANA');
```

NUME	FUNCȚIE
AVRAM ION	VÂNZATOR
BARBU DAN	VÂNZATOR
DAN ION	VÂNZATOR
MANU DAN	VÂNZATOR
VLAD ION	VÂNZATOR
SANDU ION	VÂNZATOR
CARMEN ANA	VÂNZATOR
RADU IOANA	VÂNZATOR
ALEXE IOAN	VÂNZATOR

9 records selected.

2) Să se selecteze, în modul distinct, valorile timpurilor MARCA, SALA, NUME, CODS ale angajaților care au salariul mai mare decât unul dintre subordonații superiorului cu codul 1000. Rezultatele sunt cerute ordonate descrescător, după valorile câmpului SALA.

```
SQL> SELECT DISTINCT MARCA, SALA, NUME, CODS
2        FROM SALARIATI
3        WHERE SALA> ANY
4        (SELECT SALA FROM SALARIAȚI
5        WHERE CODS=1000)
6        ORDER BY SALA DESC;
```

MARCA	SALA	NUME	CODS
2500	36500	VLAD VASILE	2500
3755	36500	DORU DAN	4000
2550	36000	FRINCU ION	2500
1000	35000	COMAN RADU	1000
8700	27500	MÂNU DAN	2500
8770	26500	CARMEN ANA	4000
8755	25700	ALEXE IOAN	2500
8760	25600	SANDU ION	4000

8650	25060	VLAD ION	1000
8600	24500	DAN ION	2500
1111	21200	AVRAM ION	1000

11 records selected.

3) Să se selecteze valorile câmpurilor MARCA, NUME, SALA, CODS ale angajaților care au salariul mai mare decât al oricărui salariat din subordinea angajatului cu codul 1000. Datele să fie ordonate descrescător după valorile câmpului MARCA.

```
SQL> SELECT MARCA, NUME, SALA, CODS
2       FROM SALARIAȚI
3       WHERE SALA>ALL
4       (SELECT SALA FROM SALARIAȚI
5       WHERE CODS=1000)
6       ORDER BY MARCA DESC;
```

MARCA	NUME	SALA	CODS
3755	DORU DAN	36500	4000
2550	FRINCU ION	36000	2500
2500	VLAD VASILE	36500	2500

3 records selected.

4) Să se selecteze câmpurile CODD, DEND, NRSAL pentru acele depozite care au numărul de salariați mai mare ca 5 şi codul cuprins în intervalul 100000 şi 130000.

```
SQL> SELECT CODD, DEND, NRSAL
2       FROM DEPOZITE
3       WHERE NRSAL>5 AND CODD IN
4       (SELECT CODD FROM DEPOZITE
5       WHERE CODD BETWEEN 100000 AND 130000);
```

CODD	DEND NRSAL
120000	ALIMENTAR 11

1 record selected.

5) Să se selecteze CODD, DEND, NRSAL pentru acele depozite care au numărul de salariați mai mare ca 5 şi codul în afara intervalului 100000 şi 130000.

```
SQL> SELECT CODD.DEND,NRSAL
2       FROM DEPOZITE
3       WHERE NRSAL>5 AND CODD NOT IN
4       (SELECT CODD FROM DEPOZITE
5       WHERE CODD BETWEEN 100000 AND 130000);
```

CODD	DEND NRSAL
140000	TEXTILE 7

1 record selected.

6) Să se selecteze următoarele informații: marca, funcția şi veniturile totale ale salariaților care au funcția şi salariul lui VLAD VASILE.

```
SQL> SELECT MARCA, NUME, SALA+VENS
        FROM SALARIAȚI
        WHERE (FUNCT, SALA)=
        (SELECT FUNCT,SALA FROM SALARIAȚI
        WHERE NUME=' VLAD VASILE ');
```

MARCA	NUME	SALA+VENS
1500	VLAD VASILE	38000
3755	DORU DAN	42000

2 records selected.

7) Să se selecteze MARCA, NUME, SALA+VENS pentru acei angajați care au funcția lui VLAD VASILE sau salariul mai mare sau egal cu cel pe care îl are RADU IOANA. Ordonarea este crescătoare după valorile câmpului NUME.

```
SQL> SELECT MARCA, NUME, SALA+VENS
        FROM SALARIAȚI
        WHERE FUNCT IN
                (SELECT FUNCT FROM SALARIAȚI
                 WHERE NUME='VLAD VASILE')
                OR SALA>=
                        (SELECT SALA FROM SALARIATI
                        WHERE NUME='RADU IOANA')
        ORDER BY NUME ;
```

NUME	FUNCT	SALA+VENS
ALEXE IOAN	VÂNZATOR	22200
AVRAM ION	VÂNZATOR	22750
BARBU DAN	VÂNZATOR	22200
CARMEN ANCA	VÂNZATOR	23456
COMANRADU	SEF DEP	37500
DAN ION	VÂNZATOR	24850
DORU DAN	SEF DEP	42000
FRINCU ION	SEF DEP	73000
MANU DAN	VÂNZATOR	30000
RADU IOANA	VÂNZATOR	23750
SANDU ION	VÂNZATOR	25600
VLAD ION	VÂNZATOR	28560
VLAD VASILE	SEF DEP	38000

13 records selected.

Dacă se doreşte evitarea afişării valorilor NULE ale câmpului sumă venituri totale (SALA+VENS), atunci se va proceda astfel:

```
SQL> SELECT MARCA, NUME, SALA+VENS
2        FROM SALARIAŢI
3        WHERE FUNCT IN
4                (SELECT FUNCT FROM SALARIAŢI
5                 WHERE NUME='VLAD VASILE')
6                OR SALA>=
7                        (SELECT SALA FROM SALARIATI
8                        WHERE NUME='RADU IOANA')
9                AND VENS IS NOT NULL
10       ORDER BY NUME ;
```

8) Să se selecteze câmpurile NUME şi CODD ale angajaților cu codul superiorului 1000 şi care au aceeaşi Funcţie cu DORU DAN. Să se afişeze numai salariații pentru care există corespondență de CODD în tabelele DEPOZITE şi SALARIAŢI. Datele să fie ordonate crescător după valorile câmpurilor CODD şi NUME.

```
SQL> SELECT NUME, DEPOZITE.CODD
2        FROM SALARIATI, DEPOZITE
3        WHERE CODS = 1000
4                AND DEPOZITE.CODD=SALARIATI.CODD
5                AND FUNCT IN
6                        (SELECT FUNCT FROM SALARIAŢI
7                        WHERE NUMEs'DORU DAN')
8 ORDER BY DEPOZITE.CODD, NUME;
```

NUME	CODD
COMAN RADU	130000

1 record selected.

9) Să se selecteze câmpurile MARCA, NUME, CODD, DEND, SALA+VENS pentru angajații care au salariul mai mare decât media salariilor realizate în depozitul din care fac ei parte.

```
SQL> SELECT MARCA, NUME, SALARIATI.CODD,
2        DEND, SALA+VENS
3        FROM SALARIATI, DEPOZITE
4        WHERE SALA>
5        (SELECT AVG(SALA) FROM SALARIAŢI
6        WHERE SALARIATI.CODD=DEPOZITE.CODD)
7        AND DEPOZITE.CODD=SALARIATI.CODD
8        ORDER BY MARCA;
```

NUME	CODD	DEND	SALA+VENS
COMAN RADU	130000	AUTO	37500

VLAD VASILE	**160000**	**SPORT**	**38000**
FRINCU ION	**160000**	**SPORT**	**73000**
DORU DAN	**130000**	**AUTO**	**42000**

6.7. Utilizarea expresiilor, funcțiilor, variabilelor sistem şi pseudo-coloanelor în selectarea datelor

Expresiile aritmetice pot fi utilizate în comenzile de selectare a datelor. Ele se construiesc cu ajutorul operatorilor aritmetici, numelor de coloane şi constantelor.

Exemple:

1) Să se selecteze câmpurile CODD, CODP, CODC, precum şi CANT*PRET denumit ca valoare totală, din tabela COMENZI, pentru toate înregistrările al căror cod de depozit este 100000.

SQL> SELECT CODD, CODP, CODC, CANT*PRET
2 FROM COMENZI, DEPOZITE
3 WHERE CODD=100000 ;

CODD	CODP	CODC	VALOARE TOTALĂ
100000	13333	121111	320000
100000	14444	121111	27000
100000	16666	121111	375000
100000	22222	121111	282000

4 records selected

2) Să se selecteze codurile produselor şi data până la care prețurile sunt admise, pentru produsele din tabela COMENZI care au prețul negociat mai mare decât prețul mediu stabilit.

SQL> SELECT DISTINCT PRETURI.CODP, DATASF
2 FROM PRETURI, COMENZI
3 WHERE COMENZI.PRET > PRETMIN+PRETMAX)/2

CODP	DATASF
14444	01-NOV-05
11111	30-AUG-05
12222	30-SEP-05
16666	01-NOV-05
13333	01-GCT-05

5 records selected.

3) Să se afişeze numele, marca, raportul VENS/SALA şi veniturile totale pentru şefii de depozite. Ordonarea datelor să fie făcută crescător după valorile raportului menționat.

SQL> SELECT NUME, MARCA, VENS/SALA, VENS+SALA

2 FROM SALARIATI
3 WHERE FUNCT='SEF DEP'
4 ORDER BY VENS/SALA DESC ;

NUME	MARCA	VENS/SALA	VENS+SALA
FRINCU ION	2550	1.0277777777777778	73000
DORU DAN	3755	.15068493150684932	42000
COMAN RADU	1000	.07142857142857143	37500
VLAD VASILE	2500	.0410958904109589	38000

4 records selected

4) Să se selecteze numele, codul depozitului, venitul total lunar şi anual prognozat din tabela SALARIAȚI, pentru vânzători. Ordonarea să fie făcută crescător după valorile câmpului NUME.

SQL> SELECT NUME,CODD, SALA+VENS, (SALA+VENS)*12
2 FROM SALARIAȚI
3 WHERE FUNCT='VÂNZATOR'
4 ORDER BY NUME;

NUME	CODD	SALA+VENS	(SALA+VENS)*12
ALEXE IOAN	160000	26500	318000
AVRAM ION	100000	22200	266400
BARBU DAN	120000	22750	273000
CARMEN ANCA	130000	26500	318000
DAN ION	160000	24850	298200
MANU DAN	160000	30000	360000
RADU IOANA	130000	23750	285000
SANDU ION	130000	25600	307200
VLAD ION	120000	28560	342720

9 records selected

O expresie aritmetică în care un operand este nul (valoarea NULL) are o valoare nulă (NULL). De aceea, de multe ori, construirea corectă a expresiilor aritmetice presupune transformarea valorii NULL într-o altă valoare, eventual nulă față de operația aritmetică (cum ar fi, de exemplu, zero la adunare, unu la înmulțire etc.).

Se presupune că în câmpul VENS s-au introdus şi valori NULL. Pentru astfel de valori, expresia SALA+VENS are valoarea NULL, indiferent ce valoare are SALA, ceea ce matematic nu este corect. De aceea, pentru exemplul anterior, se recomandă ca valoarea NULL a câmpului VENS să fie transformată în zero (fiind o sumă) înaintea evaluării expresiei aritmetice.

Exemple:

1) Să se selecteze coloanele NUME, MARCA, VENS, SALA+VENS din tabela SALARIAȚI, în condițiile în care codul superiorului este 1000.

SQL> SELECT NUME, MARCA,VENS, SALA+VENS
2 FROM SALARIATI
3 WHERE CODS= 1000 ;

NUME	**MARCA**	**VENS**	**SALA+VENS**
AVRAM ION	**1111**	**1000**	**22200**
BARBU DAN	**1222**	**2000**	**22750**
COMAN RADU	**1000**	**2500**	**36000**
VLAD ION	**2650**		

Deoarece venitul suplimentar al salariatului VLAD ION este NULL, venitul lui total (SALA+VENS) a rezultat tot NULL. Utilizînd *funcția NVL*, ca în exemplul de mai jos, VENITUL_TOTAL este acum calculat corect (pentru toți salariații).

SQL> SELECT NUME,
2 SALA+NVL(VENS,0) VENIT_TOTAL
3 FROM SALARIAȚI
4 ORDER BY NUME ;

NUME	**VENIT_TOTAL**
ALEXE IOAN	**25700**
AVRAM ION	**22200**
BARBU DAN	**22750**
CARMEN ANCA	**26500**
COMAN RADU	**35000**
DAN ION	**24850**
DORU DAN	**42000**
FRINCU ION	**73000**
MANU DAN	**30000**
RADU IOANA	**23750**
SANDU ION	**25600**
VLAD ION	**28560**
VLAD VASILE	**38000**

13 records selected.

În capul de tabel, rezultat în urma cererilor, apar numele coloanelor din baza de date. În locul acestora pot fi afişate *etichete de coloană* declarate în comenzile de definire a tabelelor**.**

Sintaxa de declarare este:

SELECT
coloana1 nume-etichetă1, coloana2 nume-etichetă2, …;

Notațiile pentru coloane cu etichete au fost utilizate şi în exemplele anterioare, cum ar fi cele pentru ”Diferența”, “Valoare Totală” sau “Diferență MIN sau MAX” etc.

Exemplu:

Să se selecteze denumirea depozitelor, codul acestora şi numărul de salariați ce îşi desfăşoară activitatea în cadrul lor, ordonate crescător după denumire.

```
SQL> SELECT DEND “PROFILUL”,
2        CODD “CODUL”,
3        NRSAL “NUMĂR DE SALARIAȚI”
4        FROM DEPOZITE
5        ORDER BY DEND;
```

PROFILUL	CODUL	NUMĂR DE SALARIAȚI
ALIMENTAR	120000	11
AUTO	130000	5
MOBILA	100000	3
SPORT	160000	4
TEXTILE	140000	7

Funcțiile aritmetice sunt utilizate în cererile de selecție.

***Exemple*:**

1) Să se afişeze codul, denumirea şi cantitatea - ridicată la pătrat, pentru produsele cu unitatea de măsură ‘BUC’.

```
SQL> SELECT CODP, CODUL,
2        DENP, DENUMIRE,
3        POWER (CANT,2)    CANT_PATRAT
4        FROM PRODUSE
5        WHERE UM='BUC'
6        ORDER BY CODP ;
```

CODUL	DENUMIRE	CANT_PATRAT
14444	SCAUN D4	1296
11111	MESE 15/20	49
12222	FOTOLIU A3	144
13333	CANAPEA A7	36

4 records selected.

2) Să se selecteze în modul distinct câmpurile CODP, DENP, CODC.DENC şi să se calculeze câmpul [(CANT*PRET)/2], rotunjit la două zecimale.

```
SQL>   SELECT DISTINCT
2        COMENZI.CODP, DENP,
3        COMENZI.CODC, DENC,
```

```
4       ROUND ( (COMENZI.CANT*COMENZI.PRET)/2,2 )
ROTUNJIRE
5       FROM COMENZI, CLIENTI, PRODUSE
6       WHERE COMENZI.CODP = PRODUSE.CODP
7       AND COMENZI.CODC = CLIENTI.CODC;
```

CODP	DENP	CODC	DENC	ROTUNJIRE
13333	CANAPEA A7	121111	UNIT-2	160000
16666	PLACAJ 2/2	121111	UNIT-2	187500
14444	SCAUN D4	121111	UNIT-2	13500

3 records selected.

Funcțiile caracter operează asupra şirurilor de caractere. Ele se utilizează pentru transformarea literelor mari în mici sau invers, extragerea unui subşir dintr-un şir, începând cu o anumită poziție, selectarea cuvintelor care se pronunță asemănător cu un şir dat etc.

Exemple:

1) Să se afişeze o situație finală prin care să fie redate câmpurile NUME şi MARCA angajatului, reunite într-un câmp comun denumit “INFORMAȚIE”, iar câmpul venituri totale anuale să fie denumit “VENIT_ANUAL”. Selecția este cerută pentru angajații cu codul superiorului egal cu 1000.

```
SQL>SELECT NUME || ‘-’ || MARCA
INFORMAȚIE
2       (SALA+VENS)*12   VENIT_ANUAL
3       FROM SALARIAȚI
4       WHERE CODS =1000;
```

INFORMAȚIE	VENIT_ANUAL
AVRAM ION - 1111	266400
BARBU DAN - 1222	273000
COMAN RADU - 1000	450000
VLAD ION - 2650	342720

4 records selected.

2) Să se selecteze câmpurile NUME şi FUNCT din tabela SALARIAȚI şi să se atribuie un cod de clasificare fiecărei funcții. Codul este format dintr-o singură cifră şi are valorile: 1 pentru vânzător, 2 pentru director, 3 pentru restul funcțiilor.

```
SQL> SELECT NUME, FUNCT,
2       DECODE(FUNCT, ‘VÂNZATOR’,l ,’DIRECTOR’,2,3)
3         CLASIFIC_FUNCT
4       FROM SALARIAȚI
5       ORDER BY FUNCT, NUME ;
```

NUME	FUNCT	CLASIFIC_FUNCT

COMAN RADU	**SEF DEP**	**3**
DORU DAN	**SEF DEP**	**3**
FRINCU ION	**SEF DEP**	**3**
VLAD VASILE	**SEF DEP**	**3**
ALEXE IOAN	**VÂNZATOR**	**1**
AVRAM ION	**VÂNZATOR**	**1**
BARBU DAN	**VÂNZATOR**	**1**
CARMEN ANCA	**VÂNZATOR**	**1**
DAN ION	**VÂNZATOR**	**1**
MANU DAN	**VÂNZATOR**	**1**
RADU IOANA	**VÂNZATOR**	**1**
SANDU ION	**VÂNZATOR**	**1**
VLAD ION	**VÂNZATOR**	**1**

13 records selected

3) Să se afişeze salariul şi veniturile suplimentare ale tuturor angajaților, cu specificarea numelui numai pentru salariații vânzători. Pentru restul angajaților să se afişeze mesajul: *** Nu interesează ***. Ordonarea să se facă după funcție, în mod descrescător.

SQL> SELECT DECODE
2 (
3 FUNCT,'VÂNZATOR', NUME, 'Nu interesează***'***
4) "NUMELE",
5 SALA, NVL (VENS, 0)
6 FROM SALARIAȚI
7 ORDER BY FUNCT DESC;

NUMELE	**SALA**	**VENS**
AVRAM ION	**21200**	**1000**
BARBU DAN	**20750**	**2000**
DAN ION	**24500**	**350**
MANU DAN	**27500**	**2500**
VLAD ION	**25060**	**3500**
SANDU ION	**25600**	**0**
CARMEN ANCA	**26500**	**0**
RADU IOANA	**20750**	**3000**
ALEXE IOAN	**25700**	**0**
***** Nu interesează *****	**35000**	**2500**
***** Nu interesează *****	**36500**	**1500**
***** Nu interesează *****	**36000**	**37000**
***** Nu interesează *****	**36500**	**5500**

13 records selected.

4) Să se afişeze primele 5 caractere din NUME, MARCA şi primul caracter din funcție, pentru toți angajații.

SQL> SELECT SUBSTR (NUME, l, 5) 5_din_Nume,
2 MARCA, Marca_Sal,
3 SUBSTR (FUNCT, 1, 1) 1_din_Funcție
4 FROM SALARIAȚI;

5_din_Nume	Marca_Sal	1_din_Funcție
AVRAM	1111	V
BARBU	1222	V
COMAN	1000	S
DAN I	3500	V
VLAD	2500	S
MANU	3700	V
FRINC	2550	S
VLAD	2650	V
DORU	3755	S
SANDU	3760	V
CARME	3770	V
RADU	1680	V
ALEXE	3755	V

13 records selected.

5) Să se selecteze şi afişeze numele, funcția şi salariul total pentru toți angajații care au numele terminat cu litera N. Situația trebuie aibă următoarea formă: Persoana_cu_Funcția SALA+VENS NUME SALARIAT - funcție

SQL> SELECT NUME || '-' || LOWER (FUNCT) Persoana_cu_Funcția,
2 SALA+VENS SAL_TOTAL
3 FROM SALARIAȚI
4 WHERE
5 UPPER (NUME) LIKE ' %N '

Persoana_cu_Funcția	SAL_TOTAL
AVRAM ION-vânzător	22200
BARBU DAN- vânzător	22750
DAN ION- vânzător	24850
MÂNU DAN- vânzător	30000
FRINCUION-sefdep	73000
VLAD ION- vânzător	28560
DORUDAN-sefdep	42000
SANDU ION- vânzător	25600
ALEXE IOAN-vânzator	

5) Să se selecteze câmpurile CODC, DENC, STR şi NR din tabela CLIENȚI, pentru clienții cu cifra 1 pe prima poziție a contului lor.

SQL> SELECT CODC, DENC, STR, NR
2 FROM CLIENȚI WHERE
3 INSTR (CONT,'1',1)=1 ;

CODC	DENC	STR	NR
121111	UNIT-1	Moşilor	104
211111	UNIT-2	Dorobanți	18

6) Să se afişeaze numele şi marca acelor angajaţi al căror nume se pronunţă asemănător cu DORU DAN.

```
SQL>  SELECT NUME, MARCA
2     FROM SALARIAŢI
3     WHERE  SOUNDEX  (NUME)  =  SOUNDEX  (‘DORU DAN’) ;
```

NUME	MARCA
DORU DAN	3755
DORU DANIEL	5565

2 records selected.

7) Să se selecteze din tabela SALARIAŢI coloanele NUME şi MARCA, pentru angajaţii a căror funcţie este asemănătoare fonetic cu şirul de caractere: 'vânzător'.

```
SQL>  SELECT NUME, MARCA
2     FROM SALARIAŢI
3     WHERE SOUNDEX (FUNCT)= SOUNDEX (‘VÂNZATOR’)
```

NUME	MARCA
AVRAM ION	1111
BARBU DAN	1222
DAN ION	3500
MANU DAN	3700

4 records selected.

8) Să se selecteze constanta “NUMELE SI CODUL CLIENŢILOR:” şi valorile câmpurilor DENC, CODC pentru clienţii din strada Calea Moşilor şi cu un cod mai mare ca 100000.

```
SQL> SELECT      "NUMELE SI CODUL CLIENŢILOR:" ,
2     DENC,      CODC
3     FROM CLIENŢI
4     WHERE STR LIKE ‘MOŞILOR%’
5     AND CODC> 100000 ;
```

NUMELE ŞI CODUL CLIENŢILOR:	DENC	CODC
NUMELE ŞI CODUL CLIENŢILOR:	UNIT-2	121111

Pentru afişarea câmpurilor de tip **dată calendaristică** sau pentru calcule în care sunt implicate aceste câmpuri, există funcţii specifice.

Exemple:

1) Să se selecteze în modul distinct codurile şi denumirile produselor, precum şi a datei până la care preţurile actuale sunt admise. Ordonarea să se facă crescător după valorile cimpului CODP.

```
SQL> SELECT DISTINCT   PRODUSE.CODP,
```

```
2        PRODUSE.DENP.DATASF
3        FROM PRETURI,PRODUSE
4        WHERE PRODUSE.CODP=PRETURI.CODP
5        ORDER BY PRETURI.CODP ;
```

CODP	DENP	DATASF
11111	MESE 15/20	30-AUG-05
12222	FOTOLIU A3	30-SEP-05
13333	CANAPEA A7	01-OCT-05
15555	BIROU C6X4	01-OCT-05
16668	PLACAJ 2/2	01-NOV-05
14444	SCAUN D4	01-NOV-05

6 records selected.

2) Să se selecteze câmpurile CODP şi DATASF scriindu-se codul pe un rând şi data pe următorul. Data va fi scrisă sub forma LL/ZZ-AA :
Se vor folosi operatorul TRUNC care asigură trecerea la rândul următor, funcția TO_CHAR (expr [fmt]) care efectuează conversia câmpului dată *expr* într-un şir de caractere în formatul specificat în *[fmt]*; comanda COLUMN care defineşte un alt format de afişare a coloanei DATASF.

```
SQL> COLUMN DATASF FORMAT A21 TRUNC
SQL> SELECT      CODP,
 2      TO_CHAR (DATASF, 'MM/DD-YY') DATASF
 3      FROM PRETURI;
```

CODP	DATASF
11111	08/30-05
12222	09/30-05
13333	10/01-05
15555	10/01-05
16666	11/01-05
14444	11/01-05

6 records selected.

3) Să se selecteze coloanele CODP, DATASF la produsele cu codul mai mare ca 13333, afişând ziua şi luna în litere iar anul cu patru cifre.

```
SQL> COLUMN DATASF FORMAT A26
SQL> SELECT CODP,
2        TO_CHAR (DATASF, 'DAY MONTH YYYY') DATASFÂRŞIT
3        FROM PRETURI
4        WHERE CODP>13333 ;
```

CODP	DATASFÂRŞIT
15555	THURSDAY OCTOBER 2005
16666	SUNDAY NOVEMBER 2005
14444	SUNDAY NOVEMBER 2005

3 records selected

4) Să se selecteze CODP, DATASF la produsele cu codul mai mare ca 13333, afişînd luna în litere şi pentru an ultimele două cifre.

SQL> SELECT CODP,
2 TO_CHAR (DATASF, 'DAY MONTH YY') DATASFÂRŞIT
3 FROM PRETURI WHERE CODP> 13333 ;

CODP	DATASFÂRŞIT	
15555	THURSDAY OCTOBER	05
16666	SUNDAY NOVEMBER	05
14444	SUNDAY NOVEMBER	05

3 records selected

5) Să se selecteze coloanele CODP, DATASF la produsele cu codul mai mare ca 13333, afişînd ziua în cifre, luna în litere şi anul cu patru cifre.

SQL> SELECT CODP,
2 TO_CHAR (DATASF, 'DD MONTH YYYY') DATASFÂRŞIT
3 FROM PRETURI
4 WHERE CODP> 13333 ;

CODP	DATASFÂRŞIT		
15555	01	OCTOBER	2005
16666	01	NOVEMBER	2005
14444	01	NOVEMBER	2005

3 records selected

6) Să se selecteze coloanele CODP, DATASF la produsele cu codul mai mare ca 13333, afişînd primele trei litere de la zi, luna în litere şi anul cu patru cifre.

SQL> SELECT CODP,
2 TO_CHAR (DATASF, 'DY MONTH YYYY') DATASFÂRŞIT
3 FROM PRETURI
4 WHERE CODP> 13333 ;

CODP	DATASFÂRŞIT		
15555	THU	OCTOBER	2005
16666	SUN	NOVEMBER	2005
14444	SUN	NOVEMBER	2005

3 records selected

7) Să se selecteze câmpurile CODP, DATASF la produsele cu codul mai mare ca 13333, afişând ziua în litere, luna în cifre şi anul cu patru cifre.

SQL>SELECT CODP,
2 TO_CHAR (DATASF, 'DAY MM YYYY') DATASFÂRŞIT
3 FROM PRETURI
4 WHERE CODP> 13333 ;

CODP	DATASFÂRŞIT
15555	THURSDAY 10 2005
16668	SUNDAY 11 2005
14444	SUNDAY 11 2005

3 records selected.

8) Să se selecteze câmpurile CODP, DATASF la produsele cu codul mai mare ca 13333, afişând primele trei litere pentru zi şi lună şi ultimele două cifre de la an.

SQL>SELECT CODP,
2 TO_CHAR (DATASF, 'DY-MON-YY') DATASFÂRŞIT
3 FROM PRETURI
4 WHERE CODP>13333 ;

CODP	DATASFÂRŞIT
15555	THU-OCT-05
16666	SUN-NOV-05
14444	SUN-NOV-05

3 records selected.

9) Să se selecteze câmpurile CODP, DATASF pentru produsele cu codul mai mare decât 13333, afişându-se primele trei litere de la zi, luna în litere şi ultimele două cifre de la an.

SQL>SELECT CODP,
2 TO_CHAR (DATASF, 'DY-MONTH-YY') DATASFÂRŞIT
3 FROM PRETURI
4 WHERE CODP>13333 ;

CODP DATA SFÂRŞIT
16556 THU-OCTOBER -05
16666 SUN-NOVEMBER -05
14444 SUN-NOVEMBER -05

3 records selected.

10) Să se selecteze coloanele CODP, DATASF pentru toate produsele, afişând ziua în cifre urmate de sufixul *-th*, primele trei litere ale lunii şi ultimele două cifre ale anului, despărţite prin *liniuţă*.

SQL>SELECT CODP,

2 TO CHAR (DATASF, 'DDth-MON-YY') DATASFÂRŞIT
3 FROM PRETURI ;

CODP	DATASFÂRŞIT
11111	30TH-AUG-05
12222	30TH-SEP- 05
13333	01ST-OCT- 05
16555	01ST-OCT- 05
16666	01ST-NOV- 05
14444	01ST-NOV- 05

6 records selected.

11) Să se selecteze din tabela PRETURI valorile câmpurilor CODP şi DATASF. Data de sfârşit (DATASF) să se prezinte însoțită de timpul intern, exprimat în diverse forme de afişare.

SQL>SELECT CODP,
2 TO_CHAR (DATASF, 'DDth-MON-YY HH:MIPM') DATASFÂRŞIT
3 FROM PRETURI ;

CODP	DATASFÂRŞIT	
11111	30TH-AUG-05	12:00AM
12222	30TH-SEP-05	12:00AM
13333	01ST-OCT-05	12:00AM
15555	01ST-OCT-05	12:00AM
16666	01ST-NOV-05	12:00AM
14444	01ST-NOV-05	12:00AM

6 records selected.

Sau :

SQL> SELECT CODP,
2 TO_CHAR (DATASF, 'DDth-MON-YY HH:MI') DATASFÂRŞIT
3 FROM PRETURI ;

CODP	DATASFÂRŞIT	
11111	30TH-AUG-05	12:00
12222	30TH-SEP-05	12:00
13333	01ST-OCT-05	12:00
15555	01ST-OCT-05	12:00
16666	01ST-NOV-05	12:00
14444	01ST-NOV-05	12:00

6 records selected.

12) Să se afişeze câmpurile CODP, DATASF şi data de sfârşit a valabilității unui preț (considerată la o distanță de 90 de zile față de DATASF).

SQL> SELECT CODP, DATASF, DATASF+90
2. FROM PRETURI;

CODP	DATASF	DATASF+90
11111	30-AUG-05	28-NOV-05
12222	30-SEP-05	29-DEC-05
13333	01-OCT-05	30-DEC-05
15555	01-OCT-05	30-DEC-05
16666	01-NOV-05	30-JAN-06
14444	01-NOV-05	30-JAN-06

6 records selected

13) Să se selecteze codul produsului, data maximă admisă de practicare a unui preț şi data curentă pentru acele produse care îndeplinesc condiția ca DATASF+10 să fie mai mare decât SYSDATE.

```
SQL>SELECT CODP, DATASF,
2         SYSDATE DATA_CURENTĂ
3         FROM PRETURI
4         WHERE DATASF+10 > SYSDATE ;
```

CODP	DATASF	DATA_CURENTĂ
12222	30-SEP-05	13-SEP-05
13333	01-OCT-05	13-SEP-05
15555	01-OCT-05	13-SEP-05
16666	01-NOV-05	13-SEP-05
14444	01-NOV-05	13-SEP-05

5 records selected.

14) Să se selecteze codul produsului, data de sfârşit, data curentă, valorile expresiilor TRUNC(DATASF+90) - TRUNC (SYSDATE) şi DATASF+90. Selecția să se realizeze pentru produsele cu codul mai mare ca 13333 şi data de sfârşit plus 90 de zile mai mare decât data curentă.

```
SQL>SELECT CODP, DATASF,
2          SYSDATE DATA_CURENTA,
3          TRUNC(DATASF+90)-TRUNC(SYSDATE)
DIFERENȚA
4          DATASF+90 DATA_90
5         FROM PRETURI
6         WHERE DATASF+90 > SYSDATE
7         AND CODP>13333 ;
```

CODP	DATASF	DATA_CURENTA	DIFERENTA	DATA_90
15555	01-OCT-05	02-OCT-05	89	30-DEC-05
16666	01-NOV-05	02-OCT-05	120	30-JAN-06
14444	01-NOV-05	02-OCT-05	120	30-JAN-06

3 records selected.

Operațiile de calcul cu data calendaristică sunt posibile în cadrul unei comenzi de selecție. Structura afişării câmpurilor rezultate se poate stabili prin comanda COLUMN.

15) Să se selecteze şi afişeze coloanele CODP, DATASF, RDATE şi RSDATE (utilizând funcțiile NEXT_DAY şi LAST_DAY).

SQL>COLUMN DATASF FORMAT A21
SQL>COLUMN RDATE FORMAT A21
SQL>COLUMN RSDATE FORMAT A20
SQL>SELECT CODP,
TO_CHAR (DATASF, 'DAY MON - YY') DATASFÂRŞIT
TO_CHAR (NEXT_DAY (DATASF+90, 'VINERI'), 'DAY MON - YY') RDATE
TO_CHAR (LAST_DAY (DATASF+90), 'DAY MON - YY') RSDATE
FROM PREȚURI ;

CODP	DATASF	RDATE	RSDATE
11111	WEDNESDAY AUG-05	VINERI NOV-05	WEDNESDAY NOV-05
12222	FRIDAY SEP-05	VINERI DEC-05	SATURDAY DEC-05

2 records selected

O operație posibil de realizat este şi cea de a aduna algebric un număr de luni la o dată calendaristică: funcția ADD_MONTHS (d,n), care adună *n* luni la data *d*.

16) Să se afişeze codul produsului, data de sfârşit şi a unui nou termen de valabilitate a unui preț dat, calculat prin adăugarea a trei luni. Să se selecteze doar produsele al căror cod este mai mic decât 13333, iar data de sfârşit este mai mare decât data curentă plus trei luni.

```
SQL>SELECT CODP, DATASF,
2       ADD_MONTHS (DATASF,3) RDATE
3       FROM PRETURI
4       WHERE
5       DATASF > SYSDATE+90
6       AND CODP<13333 ;
```

CODP	DATASF	RDATE
11111	30-AUG-05	30-NOV-05
12222	30-SEP-05	30-DEC-05

2 records selected

17) Să se selecteze codul, denumirea produselor şi data de sfârşit, pentru acele produse care îndeplinesc următoarele condiţii: data de sfârşit este cuprinsă în intervalul: 30/Aug/05 - l/Oct/05; preţurile sunt mai mari ca 30.000 u.m.. Datele să se ordoneze crescător după codul produsului.

```
SQL>SELECT DISTINCT PRODUSE.CODP, DENP, DATASF
2 FROM PRETURI, PRODUSE
3 WHERE
4 DATASF BETWEEN '30-AUG-05' AND '01-OCT-05'
5 AND PRET > 30000
6 AND PRODUSE.CODP=PRETURI.CODP
7 ORDER BY PRODUSE.CODP ;
```

CODP	DENP	DATASF
12222	FOTOLIU A3	30-SEP-05
13333	CANAPEA A7	01-OCT-05
15555	BIROU C6X4	01-OCT-05

3 records selected

6.8. Utilizarea funcţiilor de grupare şi a clauzei GROUP BY în selectarea datelor

Rezultatele obţinute în urma selectărilor pot fi grupate cu ajutorul *clauzei GROUP BY*. Secvenţa utilizată pentru această operaţie este:

SELECT …
GROUP BY tabelă.coloana1, tabelă.coloană2,…
HAVING condiţie
…;

Prin parcurgerea secvenţei se obţine câte un rând pentru înregistrările care au aceleaşi valori în coloanele specificate în clauza GROUP BY. Prezenţa clauzei HAVING determină obţinerea acelor grupuri care îndeplinesc condiţiile specificate. Este de reţinut faptul că GROUP BY şi HAVING trebuie să fie declarate după clauzele WHERE, CONNECT BY şi START WITH, în cazul cînd acestea există în comandă.

Exemple:
1) Să se selecteze din tabela PRETURI valorile din coloana DATASF şi să se contorizeze numărul apariţiilor acestora.

SQL> SELECT TO_CHAR (DATASF, ' "DATA - " DD MON YYYY')
2 DATA_MAXIMA,
3 COUNT (*) NUMAR_PRODUSE
4 FROM PRETURI
5 GROUP BY TO_CHAR (DATASF, ' "DATA - " DD MON YYYY');

DATA MAXIMA	NUMAR_PRODUSE
DATA **- 01 NOV 2005**	**2**
DATA **- 01 OCT 2005**	**2**
DATA **- 30 AUG 2005**	**1**
DATA **- 30 SEP 2005**	**1**

2) Să se selecteze şi afişeze valoarea medie zilnică a comenzilor ce trebuie onorate în perioada 01-30 Iulie 2005

SQL> SELECT AVG (CANT*PRET) MEDIA
2 FROM COMENZI
3 WHERE
4 DATAL >'01-JUL-05'AND DATAL < '30-JUL-05' ;

MEDIA
25100

3) Să se afişeze valoarea totală a salariilor şi veniturilor suplimentare pentru salariaţii cu funcţia vânzator.

SQL> SELECT SUM (SALA) TOTAL_SAL ,
2 SUM (VENS) TOTAL_VEN
3 FROM SALARIAȚI
4 WHERE FUNCT = 'VÂNZATOR';

TOTAL_SAL	TOTAL_VEN
268560	**14750**

4) Să se afişeze media anuală a veniturilor totale (SALA+VENS) pentru salariaţii cu funcţia vânzător.

SQL> SELECT AVG (SALA+VENS)*12 MEDIA_ANUALA
FROM SALARIAȚI
WHERE FUNCT='VÂNZATOR';

MEDIA_ANUALA
308147

5) Să se afişeze valoarea maximă şi minimă a salariului precum şi diferenţa max-min. Selecţia se face din tabela SALARIAŢI.

SQL> SELECT MAX (SALA) MAX_SAL,
2 MIN (SALA) MIN_SAL,
3 MAX (SALA)-MIN (SALA) DIF_MAX_MIN
4 FROM SALARIAȚI ;

MAX_SAL	MIN_SAL	DIF_MAX_MIN
45000	24250	25750

6) Să se determine lungimea maximă a şirurilor de caractere din coloana DENP a tabelei PRODUSE.

SQL>SELECT MAX (LENGTH (DENP)) LUNG_MAX_DENP
2 FROM PRODUSE ;

LUNG_MAX_DENP
10

7) Să se selecteze coloanele NUME, PUNCT, SALA+VENS din tabela SALARIAȚI pentru angajații care au salariul egal cu salariul maxim.

SQL> SELECT NUME, FUNCT, SALA+VENS
2 FROM SALARIAȚI
3 WHERE SALA =
4 (SELECT MAX (SALA) FROM SALARIAȚI) ;

NUME	FUNCT	SALA+VENS
ION ION	DIRECTOR	85000

1 record selected

8) Să se selecteze coloanele DENP, CODP, CODD din tabela PRODUSE pentru care stocul existent este mai mare sau egal cu cantitatea comandată.

SQL> SELECT DENP, CODP, CODD
2 FROM PRODUSE
3 WHERE STOC >=
4 (SELECT SUM (CANT) FROM COMENZI);

DENP	CODP	CODD
PLACAJ 2/2	166666	100000
SCAUN D4	144444	100000
CANAPEA A7	133333	100000

9) Să se afişeze numărul de valori nenule înregistrate în coloana VENS din tabela SALARIAȚI.

SQL> SELECT COUNT (VENS)
2 FROM SALARIAȚI ;

COUNT (VENS)
11

10) Să se selecteze din tabela SALARIAȚI coloanele NUME, FUNCT, SALA+VENS pentru şefii de depozite care au salariul mai mare sau egal cu jumătate din salariul maxim.

```
SQL>SELECT NUME, FUNCT, SALA+NVL (VENS,0)
2       FROM SALARIAȚI
3       WHERE SALA >=
4       (SELECT MAX (SALA)/2 FROM SALARIAȚI)
5       AND FUNCT= 'SEF DEP' ;
```

NUME	FUNCT	SALA+NVL(VENS,0)
COMAN RADU	SEF DEP	37500
VLAD VASILE	SEF DEP	38000
FRINCU ION	SEF DEP	73000
DORU DAN	SEF DEP	42000
PAUL STEFAN	SEF DEP	40600

11) Să se afişeze numărul de produse distincte care au ca unitate de măsură 'BUC'

```
SQL>SELECT COUNT(DISTINCT CODP) NR_PROD
2       FROM PRODUSE
3       WHERE UM = 'BUC';
```

NR_PROD
5

12) Să se afişeze numărul de subordonați ai salariatului cu marca 2500.

```
SQL>  SELECT COUNT(*) NR_SUBORD
FROM SALARIAȚI
WHERE CODS=2500;
```

NR_SUBORD
3

13) Să se afişeze valoarea medie a salariilor şi valoare medie a veniturilor suplimentare pentru fiecare depozit fie utilizînd comanda SELECT de mai multe ori, fie clauza GROUP BY o singură dată.

Folosind comanda SELECT:
Pentru salariații depozitului 100000 (codd=100000)

```
SQL> SELECT AVG(SALA), AVG(VENS)
FROM SALARIAȚI
WHERE CODD = 100000;
```

AVG(SALA)   AVG(VENS)

33100	20500

Pentru salariații depozitului 120000 (codd=120000)

```
SQL>SELECT AVG(SALA), AVG(VENS)
2 FROM SALARIAȚI
3 WHERE CODD = 120000;
```

AVG(SALA)	AVG(VENS)
2402.5	1975

Pentru salariații depozitului 130000 (codd=130000)

```
SQL> SELECT AVG(SALA), AVG(VENS)
2 FROM SALARIAȚI
3 WHERE CODD = 130000;
```

AVG(SALA)	AVG(VENS)
28870	2750

Pentru salariații depozitului 160000 (codd=160000)

```
SQL> SELECT AVG(SALA), AVG(VENS)
2 FROM SALARIAȚI
3 WHERE CODD = 160000;
```

AVG(SALA)	AVG(VENS)
30866.7	9390

Folosind clauza GROUP BY:

```
SQL> SELECT CODD, AVG(SALA), AVG(VENS)
2 FROM SALARIAȚI
3 WHERE CODD = 160000;
4 GROUP BY CODD ;
```

CODD	AVG(SALA)	AVG(VENS)
100000	33100	20500
120000	24202.5	1975
130000	28870	2750
160000	30866	9390

14) Să se calculeze media salariului pentru fiecare grup de angajați care au acelaşi superior.

```
SQL> SELECT CODS, AVG(SALA)
2       FROM SALARIAȚI
3       GROUP BY CODS ;
```

CODS	AVG (SALA)
1000	22336.7
2500	27083,3
2550	25500
3755	26600
4000	26050
7000	35750
8000	4500

15) Să se calculeze media salariului anual prognozat pentru salariații care au acelaşi superior şi nu au funcția de şef depozit.

```
SQL>  SELECT CODS,AVG(SALA)*12 MEDIE_SAL_ANUAL
2         FROM SALARIAȚI
3         WHERE FUNCT NOT IN ('SEF DEP')
4         GROUP BY CODS ;
```

CODS	MEDIE_SAL_ANUAL
1000	268040
2500	271500
2550	306000
3755	319200
4000	312600
8000	540000

16) Să se calculeze şi afişeze valoarea medie a comenzilor pentru fiecare depozit.

```
SQL> SELECT AVG (CANT*PRET) VAL_MEDIE,
2         DEPOZITE.CODD    COD_DEP
3         FROM COMENZI, DEPOZITE
4         WHERE COMENZI.CODD=DEPOZITE.CODD
5         GROUP BY DEPOZITE.CODD ;
```

VAL_MEDIE	COD_DEP
251000	100000

17) Să se calculeze şi afişeze valoarea medie a comenzilor pe produse.

```
SQL> SELECT  AVG (CANT*PRET) VAL_MEDIE,
2         PRODUSE.CODP     COD_PROD
3         FROM PRODUSE, COMENZI
4         WHERE COMENZI.CODP=PRODUSE.CODP
5         GROUP BY PRODUSE.CODP ;
```

VAL_MEDIE	COD_PROD
320000	133333
27000	144444
375000	166666

18) Să se determine media salariului anual pentru fiecare funcție dacă există mai mult de doi salariați angajați pe aceeaşi funcție. Se vor afişa funcția, numărul de salariați cu uncția respectivă şi media calculată.

```
SQL> SELECT FUNCT FUNCȚIE,
2       COUNT(*) NR_SAL,
3       AVG(SALA)*12 MEDIE_SAL
4       FROM SALARIAȚI
5       GROUP BY FUNCT
6       HAVING COUNT(*)>2 ;
```

FUNCȚIE	NR_SAL	MEDIE_SAL
SEF DEP	5	429600
VÂNZATOR	11	292975

2 records selected.

19) Să se selecteze codurile superiorilor (CODS) care au doi sau mai mulți subordonați.

```
SQL>  SELECT CODS
2       FROM SAIARIATI
3       WHERE FUNCT='VÂNZATOR'
4       GROUP BY CODS
5       HAVING COUNT(*)>=2 ;
```

CODS
1000
2500
2550
3755
4000

5 records selected.

20) Să se selecteze codurile superiorilor care au media veniturilor suplimentare ale subordonaților mai mare decât 10% din salariul mediu. Se vor afişa codurile superiorilor şi media veniturilor totale anuale ale subordonaților.

```
SQL> SELECT CODS,
2       AVG (SALA+NVL (VENS,0) )*12 MED_VEN_TOT_AN
3       FROM SALARIAȚI
4       GROUP BY CODS
5       HAVING AVG (VENS) > AVG (SALA)*0.10;
```

CODS	MED_VEN_TOT_AN
2500	486400
7000	474300
8000	1020000

21) Să se selecteze funcțiile pentru care salariile medii sunt mai mari decât salariul mediu al unui vînzător. Se vor afişa funcțiile şi salariile medii pentru fiecare funcție.

```
SQL>  SELECT FUNCT, AVG(SALA) MEDIE_SAL
2         FROM SALARIAȚI
3         GROUP BY FUNCT
4         HAVING AVG (SALA) >
5         (SELECT AVG(SALA) FROM SALARIAȚI
6         WHERE FUNCT= 'VÂNZATOR' );
```

FUNCT	MEDIE_SAL
DIRECTOR	45000
SEF DEP	35800

22) Să se afişeze numărul salariaților din depozitul cu codul 100000 şi să se determine câți dintre ei au venituri suplimentare.

```
SQL>  SELECT COUNT (*) NR_SALARIAȚI_DEP_100000,
2         COUNT(VENS) NR_SAL_VEN_SUPL
3         FROM SALARIAȚI
4         WHERE CODS=100000 ;
```

NR_SALARIAȚI_DEP_100000	NR_SAL_VEN_SUPL
4	3

23) Să se determine numărul salariaților din depozitul cu codul 100000 precum şi suma şi media veniturilor lor suplimentare.

```
SQL> SELECT SUM (NVL (VENS,0)) SUMA_VEN_SUPL,
2         COUNT (NVL (VENS,0)) NR_SAL,
3         AVG (NVL (VENS,0)) MEDIA_VEN_SUPL
4         FROM SALARIAȚI
5         WHERE CODS=100000 ;
```

SUMA_VEN_SUPL	NR_SAL	MEDIA_VEN_SUPL
9000	4	2250

24) Pentru a afla numărul de angajați şi salariul mediu anual în toate combinațiile posibile de departamente şi funcții, se va folosi următoarea cerere:

SELECT DECODE(GROUPING(nume_dept), 1, 'Toate Departamentele', nume_dept) AS nume_dept,

DECODE(GROUPING(functia), 1, 'Toate Functiile', functia) AS functia,
COUNT(*) "Total Angajati",
AVG(sal) * 12 "Sal Mediu"
FROM ang, dept
WHERE dept.nume_dept = emp.nume_dept
GROUP BY CUBE (nume_dept, functia);

Nume dept.	functia	total angajati	sal mediu
CONTABILITATE	FUNCTIONAR	1	15600
CONTABILITATE	MANAGER	1	29400
CONTABILITATE	PRESEDINTE	1	60000
CONTABILITATE	Toate Functiile	3	35000
CERCETARE	ANALYST	2	36000
CERCETARE	FUNCTIONAR	2	11400
CERCETARE	MANAGER	1	35700
CERCETARE	Toate Functiile	5	26100
VANZARI	FUNCTIONAR	1	11400
VANZARI	MANAGER	1	34200
VANZARI	VANZATOR	4	16800
VANZARI	Toate Functiile	6	18800
Toate Departamentele	ANALYST	2	36000
Toate Departamentele	FUNCTIONAR	4	12450
Toate Departamentele	MANAGER	3	33100
Toate Departamentele	PRESEDINTE	1	60000
Toate Departamentele	VANZATOR	4	16800
Toate Departamentele	Toate Functiile	14	24878.5714

6.9. Operații pe tabele structurate arborescent

Limbajul SQL*Plus permite explorarea *structurilor arborescente* existente în baza de date. Operația se realizează cu ajutorul clauzelor START WITH şi CONNECT BY din comanda SELECT.

```
SELECT …
        FROM …
        CONNECT BY [PRIOR] col1 = [PRIOR] col2
        START WITH col = valoare
…;
```

Exemple:

1) Să se afişeze câmpurile NUME, FUNCT, CODS, MARCA din tabela SALARIAȚI. Datele să fie ordonate crescător după codul superiorului.

SQL>SELECT NUME, FUNCT, CODS, MARCA
FROM SALARIATI
ORDER BY CODS;

NUME	FUNCT	CODS	MARCA
AVRAM ION	VÂNZĂTOR	1000	1111
BARBU DAN	VÂNZĂTOR	1000	1222
COMAN RADU	SEF DEP	7000	1000
VLAD ION	VÂNZĂTOR	1000	2650
AILENEI FLORIN	VÂNZĂTOR	2500	2553
DAN ION	VÂNZĂTOR	2500	3500
DARIAN GEO	VÂNZĂTOR	2500	2554
FRINCU ION	SEF DEP	2500	2550
RADU ION	VÂNZĂTOR	2500	1680
VLAD VASILE	SEF DEP	7000	2500
ALEXE IOAN	VÂNZĂTOR	3755	3759
MANU DAN	VÂNZĂTOR	3755	3700
DORU DAN	SEF DEP	7000	3755
CARMEN ANCA	VÂNZĂTOR	4000	3770
PAUL ŞTEFAN	SEF DEP	7000	4000
SANDU ION	VÂNZĂTOR	4000	3760
ION ION	DIRECTOR	7000	7000

2) Să se selecteze NUMELE, MARCA, codul superiorului şi codul depozitului pentru subordonații direcți şi indirecți ai salariatului cu numele DORU DAN.

SQL> SELECT NUME, MARCA, CODS, CODD
2 FROM SALARIATI
3 CONNECT BY PRIOR MARCA = CODS
4 START WITH NUME = 'DORU DAN';

NUME	MARCA	CODS	CODD
DORU DAN	3755	7000	130000
MANU DAN	3700	3755	160000
ALEXE IOAN	3759	3755	160000

3 records selected

3) Să se afişeze câmpurile MARCA, NUME, FUNCT, CODS, CODD din tabela SALARIAȚI, ordonate crescător după marca şi codul superiorului, pentru subordonații direcți şi indirecți ai salariatului cu numele VLAD VASILE.

SQL> SELECT MARCA, NUME, FUNCT, CODS, CODD
2 FROM SALARIATI
3 CONNECT BY PRIOR MARCA = CODS
4 START WITH NUME = 'VLAD VASILE'

5 ORDER BY MARCA, CODS;

MARCA	NUME	FUNCT	CODS	CODD
1680	RADU ION	VÂNZĂTOR	2500	2553
2500	VLAD VASILE	SEF DEP	7000	3500
2550	FRINCU ION	SEF DEP	2500	2554
2553	AILENEI FLORIN	VÂNZĂTOR	2550	2550
2554	DARIAN GEOR	VÂNZĂTOR	2550	2500
2556	DAN ION	VÂNZĂTOR	2500	2500
7000	ION ION	DIRECTOR	7000	7000

7 records selected.

4) Să se afişeze câmpurile MARCA, NUME, FUNCT, CODS, CODD din. tabela SALARIAȚI, ordonate crescător după marca.

SQL> SELECT MARCA,NUME,FUNCT,CODS,CODD
2 FROM SALARIAȚI
3 ORDER BY MARCA;

MARCA	NUME	FUNCT	CODS	CODD
1000	COMAN RADU	SEF DEP	7000	130000
1111	AVRAM ION	VINZATOR	1000	100000
1222	BARBU DAN	VINZATOR	1000	120000
1680	RADU ION	VINZATOR	2500	130000
2500	VLAD VASILE	SEF DEP	7000	160000
2550	FRINCU ION	SEF DEP	2500	160000
2553	AILENEI FLORIN	VINZATOR	2550	120000
2554	DARIAN GEO	VINZATOR	2550	120000
2650	VLAD ION	VINZATOR	1000	120000
3500	DAN ION	VINZATOR	2500	120000
3700	MÂNU DAN	VINZATOR	3755	160000
3755	DORU DAN	SEF DEP	7000	130000
3759	ALEXE IOAN	VINZATOR	3755	160000
3760	SANDU ION	VINZATOR	4000	130000
3770	CARMEN ANA	VINZATOR	4000	130000
4000	PAUL ŞTEFAN	SEF DEP	7000	160000
7000	ION ION	DIRECTOR	7000	100000

17 records selected.

5) Să se selecteze numele, marca, codul superiorului, codul depozitului pentru subordonații direcți şi indirecți ai salariatului cu numele DORU DAN.

SQL> SELECT NUME,MARCA,CODS,CODD
2 FROM SALARIAȚI
3 CONNECT BY PRIOR MARCA=CODS
4 START WITH NUME='DORU DAN';

NUME	MARCA	CODS	CODD
DORU DAN	3755	7000	130000

MÂNU DAN	**3700**	**3755**	**160000**
ALEXE IOAN	**3759**	**3755**	**160000**

3 records selected.

6) Să se afişeze MARCA, NUME, FUNCT, CODS şi CODD ordonate crescător după marcă şi cod superior pentru subordonații direcți şi indirecți ai salariatului cu numele VLAD VASILE.

SQL> SELECT MARCA,NUMEJFUNCT,CODS,CODD
2 FROM SALARIAȚI
3 CONNECT BY PRIOR MARCA=CODS
4 START WITH NUME='VLAD VASILE'
5 ORDER BY MARCA.CODS

MARCA	NUME	FUNCT	CODS	CODD
1680	RADU ION	VINZATOR	2500	130000
2500	VLAD VASILE	SEF DEP	7000	160000
2550	FRINCU ION	SEF DEP	2500	160000
2553	AILENEI FLORIN	VINZATOR	2550	120000
2554	DARIAN GEO	VINZATOR	2550	120000
3500	DAN ION	VINZATOR	2500	160000

6 records selected.

7) Să se afişeze câmpurile NUME şi MARCA pentru subordonații direcți şi indirecți ai salariatului ION ION precum şi nivelul lor de subordonare.

SQL> SELECT LEVELJWME.MARCA
2 FROM SALARIAȚI
3 CONNEOT BY PRIOR MARCA=CODS
4 START WITH NUME='ION ION';

LEVEL	NUME	MARCA
1	ION ION	7000
2	COMAN RADU	1000
3	AVRAM ION	1111
3	BARBU DAN	1222
3	VLAD ION	2650
2	VLAD VASILE	2500
3	FRINCU ION	2550
4	AILENEI FLORIN	2553
4	DARIAN GEO	2554
3	RADU ION	1680
3	DAN ION	3500
2	DORU DAN	3755
3	MÂNU DAN	3700
3	ALEXE IOAN	3759
2	PAUL ŞTEFAN	4000
3	SANDU ION	3760
3	CARMEN ANA	3770

8) Să se selecteze câmpurile LEVEL, NUME şi MARCA apartinând subordonaţilor lui ION ION. Nivelul de subordonare va fi pus în evidenţă şi prin afişarea deplasată a numelor salariaţilor subordonaţi faţă de numele superiorului corespunzător.

SQL> COLUMN ORG.CHART FORMAT A21
SQL> SELECT LEVEL,
2 LPADC (‘ ’,LEVEL*2) || NUME NUME_SI_PRENUME,
3 MARCA
4 FROM SALARIAŢI
5 CONNECTBY PRIOR MARCA=CODS
6 START WITH NUME='ION ION»;

LEVEL	NUME.SI.PRENUME	MARCA
1	ION ION	7000
2	COMAN RADU	1000
3	AVRAM ION	1111
3	BARBU DAN	1222
3	VLAD ION	2650
2	VLAD VASILE	2500
3	FRINCU ION	2550
4	AILENEI FLORIN	2553
4	DARIAN GEO	2554
3	RADU ION	1680
3	DAN ION	3500
2	DORU DAN	3755
3	MÂNU DAN	3700
3	ALEXEIOAN	3759
2	PAUL ŞTEFAN	4000
3	SANDU ION	3760
3	CARMEN ANA	3770

17 records selected.

9) Să se afişeze câmpurile LEVEL, NUME, MARCA apartinând subordonaţilor salariaţilor PAUL ŞTEFAN şi COMAN RADU, punând în evidenţă, prin scriere decalată, modul de subordonare.

SQL> SELECT LEVEL,
LPAD(','LEVEL*2) || NUME
3 NUME_SI_PRENUME,MARCA
4 FROM SALARIAŢI
5 CONNECT BY PRIOR MARCA=CODS
6START WITH NUME='PAUL ŞTEFAN’ 7 OR NUME='COMAN RADU’

LEVEL	NUME	MARCA
1	COMAN RADU	1000
2	AVRAM ION	1111
2	BARBU DAN	1222
2	VLAD ION	2650

1	PAUL ŞTEFAN	4000
2	SANDU ION	3760
2	CARMEN ANA	3770

7 records selected.

10) Să se afişeze câmpurile LEVEL, NUME şi MARCA, in structură arborescentă, apartinând salariaților din subordinea celor cu aceeaşi funcție cu a salariatului COMAN RADU.

SQL> SELECT LEVEL,LPADC ',LEVEL*2) || NUME
2 NUME_SI_PRENUME, MARCA
3 FROM SALARIAȚI
4 CONNECT BY PRIOR MARCA=CODS
5 START WITH FUNCT IN
6 (SELECT FUNCT FROM SALARIAȚI
7 WHERE NUMEs"COMAN RADU");

LEVEL	NUME_SI_PRENUME	MARCA
2	COMAN RADU	1000
3	AVRAM ION	1111
3	BARBU DAN	1222
3	VLAD ION	2650
2	VLAD VASILE	2500
3	FR1NCU ION	2550
4	AILENEI FLORIN	2553
4	DARIAN GEO	2554
3	RADU ION	1680
3	DAN ION	3500
2	DORU DAN	3755
3	MÂNU DAN	3700
3	ALEXEIOAN	3759
2	PAUL ŞTEFAN	4000
3	SANDU ION	3760
3	CARMEN ANA	3770

11) Să se selecteze câmpurile LEVEL, NUME, MARCA, FUNCT, CODD aparținând subordonaților angajaților care lucrează în acelaşi depozit cu RADU ION.

SQL> SELECT LEVEL,
2 LPADC (` `,LEVEL*2) || NUME
3 NUME_SI_PRENUME,
4 MARCA,FUNCT,CODD
5 FROM SALARIAȚI
6 CONNECT BY PRIOR MARCA=CODS
7 START WITH CODD IN
8 (SELECT CODD
9 FROM SALARIAȚI
10 WHERE NUME='RADU ION')

LEVEL	NUME_SI_PRENUME	MARCA	FUNCT	CODD
1	RADU ION	1680	VINZATOR	130000
1	COMAN RADU	1000	SEF DEP	130000
2	AVRAM ION	1111	VINZATOR	100000
2	BARBU DAN	1222	VINZATOR	120000
2	VLAD ION	2650	VINZATOR	120000
1	DORU DAN	3755	SEF DEP	130000
2	MÂNU DAN	3700	VINZATOR	160000
2	ALEXE IOAN	3759	VINZATOR	160000
1	CARMEN ANA	3770	VINZATOR	130000
1	SANDU ION	3760	VINZATOR	130000

10 records selected.

12) Să se selecteze câmpurile NUME, MARCA, FUNCT, CODD aparținând superiorilor salariatului VLAD ION.

SQL> SELECT NUME,MARCA,CODS,FUNCT,CODD
2 FROM SALARIAȚI
3 CONNECT BY MARCA=PRIOR CODS
4 START WITH NUME=`VLAD ION`;

NUME	MARCA	CODS	FUNCT	CODD
VLAD ION	2650	1000	VINZATOR	120000
COMAN RADU 1000	7000	SEF	DEP	130000
ION ION	7000	8000	DIRECTOR	100000

3 records selected.

13) Să se afişeze câmpurile NUME, MARCA, FUNCT aparținând subordonaților salariatului VLAD VASILE, mai puțin datele salariatului AILENEI FLORIN.

SQL> SELECT NUME,MARCA,FUNCT
2 FROM SALARIATI
3 WHERE NUME !='AILENEI FLORIN'
4 CONNECT BY PRIOR MARCA=CODS
5 START WITH NUME='VLAD VASILE';

NUME	MARCA	FUNCT
VLAD VASILE	2500	SEF DEP
FRINCU ION	2550	SEF DEP
DARIAN GEO	2554	VINZATOR
RADU ION	1680	VINZATOR
DAN ION	3500	VINZATOR

14) Să se afişeze o serie de date despre subordonații salariatului ION ION, mai puțin datele despre VLAD VASILE şi despre salariații din subordinea lui.

SQL> SELECT MARCA,LEVEL,NUME,PUNCT,CODS
2 FROM SALARIAȚI

```
 3        CONNECT BY PRIOR MARCA=CODS
 4        AND NUME <> 'VLAD VASILE'
 5        START WITH NUME='ION ION';
```

MARCA	LEVEL	NUME	FUNCT	CODS
7000	1	ION ION	DIRECTOR	8000
1000	2	COMAN RADU	SEF DEP	7000
1111	3	AVRAM ION	VINZATOR	1000
1222	3	BARBU DAN	VINZATOR	1000
2650	3	VLAD ION	VINZATOR	1000
3755	2	DORU DAN	SEF DEP	7000
3700	3	MÂNU DAN	VINZATOR	3755
3759	3	ALEXE IOAN	VINZATOR	3755
4000	2	PAUL ŞTEFAN	SEF DEP	7000
3760	3	SANDU ION	VINZATOR	4000
3770	3	CARMEN ANA	VINZATOR	4000

11 records selected

15) Să se afişeze o serie de date ale salariaţilor subordonaţi lui ION ION, mai puţin datele salariaţilor COMAN RADU şi VLAD VASILE precum şi ale subordonaţilor lui VLAD VASILE.

```
SQL> SELECT MARCA,NUME,LEVEL,FUNCT,
2 CODS.CODD
3 FROM SALARIAŢI
4 WHERE NUME <> 'COMAN RADU'
5 CONNECT BY PRIOR MARCA=CODS
6 AND NUME != 'VLAD VASILE'
7 START WITH NUME='ION ION'
```

MARCA	NUME	LEVEL	FUNCT	CODS
7000	ION ION	1DIRECTOR	8000	100000
1111	AVRAM ION	3 VINZATOR	1000	100000
1222	BARBU DAN	3 VINZATOR	1000	120000
2650	VLAD ION	3 VINZATOR	1000	120000
3755	DORU DAN	2 SEF DEP	1000	130000
3700	MÂNU DAN	3 VINZATOR	3755	160000
3759	ALEXE IOAN	3 VINZATOR	3755	160000
4000	PAUL ŞTEFAN	2 SEF DEP	7000	160000
3760	SANDU ION	3 VINZATOR	4000	130000
3770	CARMEN ANA	3 VINZATOR	4000	130000

10 records selected.

16) Să se afişeze datele salariaţilor subordonaţi direct salariatului ION ION.

```
SQL> SELECT MAJRCA,NUME,LEVEL,
2        FUNCT,CODS,CODD
```

3 FROM SALARIAȚI
4 WHERE LEVEL=2
5 CONNECT BY PRIOR MARCA=CODS
6 START WITH NUME='ION ION'

MARCA	NUME	LEVEL	FUNCT	CODS	CODD
1000	COMAN RADU	2	SEF DEP	7000	130000
2500	VLAD VASILE	2	SEF DEP	7000	160000
3755	DORU DAN	2	SEF DEP	7000	130000
4000	PAUL ŞTEFAN	2	SEF DEP	7000	160000

4 records selected.

CAPITOLUL 7. APLICATII INFORMATICE UTILIZAND LIMBAJUL SQL

7.1. Aplicație informatică pentru activitatea de salarizare

1) Folosindu-se instrucțiunile SQL, să se creeze tabelele DatePers, DateSal, Impozitar, Pontaj, SporVechime, Taxe, Deduceri.

```
CREATE TABLE DatePers
(
        codang          number (5) primary key,
        nume            varchar2 (35),
        cnp             varchar2 (13),
        datan           date,
        adresa          varchar2 (30),
        localitate      varchar2 (15),
        telefon         varchar2 (12)
);

CREATE TABLE DateSal
(
        codang          number(5) references DatePers (codang),
        functia         varchar2 (10),
        salbaza         number (15),
        persintr        number (2),
        vechime         number (3),
        codsef number (5) references DatePers(codang)
);

CREATE TABLE Impozitar
(
        linie           number (5) primary key,
        dela            number (15),
        panala          number (15),
        suma            number (15),
        procent         number (3)
);

CREATE TABLE Pontaj
(
```

```
	codang		number (5) references DatePers(codang),
	luna		number (3),
	zilelucr	number(3),
	orezi		number(3),
	zileco		number(3),
	zilecm		number(3),
	orelucrate	number(4),
	constraint pk primary key(codang,luna)
);

CREATE TABLE SporVechime
(
	nr		number (3) primary key,
	dela		number (3),
	panala		number (3),
	procent		number (3)
);

CREATE TABLE Taxe
(
	den		varchar2 (10) primary key,
	procent		number (2),
	cotamax		number (15)
);

CREATE TABLE Deduceri
(
	den		varchar2 (30) primary key,
	cotasuma	number(15),
	cotaproc	number(2),
	cotamax		number(15)
);
```

2) Să se încarce cu date tabelele create.

```
SQL>	DELETE FROM DatePers;
	INSERT INTO DatePers VALUES
	(100, 'Ion Ion', '1234567890100', '10-JAN-1970', 'Mangaliei 100',
	'Constanta', '0722123456');
	INSERT INTO DatePers VALUES
```

```
(200, 'Popescu Ion', '1234567890200', '10-FEB-1975', 'Tomis 232',
'Constanta', '0744123456');
INSERT INTO DatePers VALUES
(300, 'Ionescu Gheorghe', '1234567890300', '10-MAR-1980', 'Ferdinand
48', 'Mangalia', '0788123456');


SQL> DELETE FROM DateSal;
INSERT INTO DateSal VALUES (100, 'Ec', 5000000, 1, 10, 300);
INSERT INTO DateSal VALUES (200, 'Inginer', 6500000, 2, 5, 300);
INSERT INTO DateSal VALUES (300, 'Director', 15000000, 0, 15, null);

SQL> DELETE FROM Impozitar;
INSERT INTO Impozitar VALUES (1, 0, 2100000, 0, 18);
INSERT INTO Impozitar VALUES (2, 2100001, 5200000, 378000,23);
INSERT INTO Impozitar VALUES (3, 5200001, 8300000, 1091000,28);
INSERT INTO Impozitar VALUES (4, 8300001, 11600000, 1959000,34);
INSERT INTO Impozitar VALUES (5, 11600001, 99999999999, 3081000,
40);

SQL> DELETE FROM Pontaj;
INSERT INTO Pontaj VALUES (100, '1', 22, 8, 3, 0, 170);
INSERT INTO Pontaj VALUES (200, '1', 30, 8, 5, 10, 200);
INSERT INTO Pontaj VALUES (300, '1', 22, 8, 0, 0, 176);


SQL> DELETE FROM SporVechime;
INSERT INTO SporVechime VALUES (1, 0, 3, 5);
INSERT INTO SporVechime VALUES (2, 4, 10, 10);
INSERT INTO SporVechime VALUES (3, 11, 20, 15);
INSERT INTO SporVechime VALUES (4, 21, 40, 20);

SQL> DELETE FROM Taxe;
INSERT INTO Taxe VALUES ('CASS', 6.5, 999999999999999);
INSERT INTO Taxe VALUES ('CAS', 9.5, 15000000);
INSERT INTO Taxe VALUES ('Somaj', 1, 999999999999999);


SQL> DELETE FROM Deduceri;
INSERT INTO Deduceri VALUES ('Deducere de baza', 1800000, 0,
1800000);
```

INSERT INTO Deduceri VALUES ('Deducere suplimentara', 0, 0.5, 3600000);
INSERT INTO Deduceri VALUES ('Chelt profesionale', 0, 15, 270000);

Interogarea tabelelor bazei de date

1) Să se afişeze informaţiile despre angajaţii firmei.
SQL> SELECT * FROM DatePers;

CODANG	NUME	CNP	DataN	ADRESA	LOCALITATE	TELEFON
100	Ion Ion	1234567890100	10-JAN-1970	Mangaliei 100	Constanta	0722123456
200	Popescu Ion	1234567890200	10-FEB-1975	Tomis 232	Constanta	0744123456
300	Ionescu Gheorghe	1234567890300	10-MAR-1980	Ferdinand 48	Mangalia	0788123456

2) Să se selecteze toţi angajaţii din Constanţa.
SQL> SELECT * FROM DatePers
WHERE localitate ='Constanta';

CODANG	NUME	CNP	DataN	ADRESA	LOCALITATE	TELEFON
100	Ion Ion	1234567890100	10-JAN-1970	Mangaliei 100	Constanta	0722123456
200	Popescu Ion	1234567890200	10-FEB-1975	Tomis 232	Constanta	0744123456

3) Să se afişeze numele tuturor angajaţilor care sunt din localităţile a căror nume începe cu litera M.

SQL> SELECT nume, localitate
FROM DatePers
WHERE localitate LIKE 'M%';

CODANG	NUME	CNP	DataN	ADRESA	LOCALITATE	TELEFON
300	Ionescu Gheorghe	1234567890300	10-MAR-1980	Ferdinand 48	Mangalia	0788123456

4) Să se afişeze codul şi salariile angajaţilor care au salariul de bază între 6000000 şi 7000000

SQL> SELECT codang, salbaza
FROM DateSal
WHERE salbaza BETWEEN 6000000 AND 7000000;

CODANG	SALBAZA
200	500000

5) Să se afişeze codul angajatului cu vechime de 10 şi respectiv 15 ani

```
SQL>  SELECT codang, vechime
      FROM DateSal
      WHERE vechime IN (10,15);
```

```
CODANG          VECHIME
-------------------------------------
   100             10
   300             15
```

6) Să se afişeze impozitarul în formatul în care apare în Monitorul Oficial

```
SQL>  SELECT      dela || ' - ' || panala ||' '|| suma || ' + ' ||
                  procent || ' % pentru ceea ce depaseste ' || dela
      FROM Impozitar;
```

```
 DELA ||'-'|| PANALA ||''||  SUMA ||  '+' || PROCENT ||'%PENTRU CEEA CE DEPASESTE'
-------------------------------------------------------------------------------------------------------
 0 - 2100000                           0  +      18%      pentru ceea ce depaseste          0
 2100001 - 5200000               378000  +       23%      pentru ceea ce depaseste 2100001
5200001 - 8300000               1091000  +       28%      pentru ceea ce depaseste 5200001
8300001 - 11600000              1959000  +       34%      pentru ceea ce depaseste 8300001
11600001 - 999999999999         3081000  +       40%      pentru ceea ce depaseste 11600001
```

7) Să se afişeze, concatenat, codul angajatului şi luna din tabela Pontaj. Pentru şirul astfel creat să se afişeze lungimea sa.

```
SQL>  SELECT CONCAT (codang, luna),  ANGAJAT_LUNA
      LENGTH (concat (codang,luna))    LUNGIME_SIR
      FROM Pontaj;
```

```
ANGAJAT_LUNA   LUNGIME_SIR
-----------------------------------------------
1001                         4
2001                         4
3001                         4
```

8) Să se afişeze valoarea 41000/32000 rotunjită la 2 şi, respectiv 3 zecimale

```
SQL>  SELECT      ROUND (41000/32000, 2)    2_ZECIMALE,
                  ROUND (41000/32000, 3)    3_ZECIMALE
                  FROM DUAL;
```

```
2_ZECIMALE       3_ZECIMALE
------------------- -------------------------
         1.28            1.281
```

9) Să se afişeze angajații care au cuvântul “Ion” în nume (nume şi prenume) împreună cu vârsta acestora. Vârsta se va afişa în două moduri: rotunjită în ani şi în ani cu luni.

SQL> SELECT nume,
ROUND ((sysdate-datan)/365, 0) ANI,
ROUND ((sysdate-datan)/365, 1) ANI_CU_LUNI
FROM DatePers
WHERE nume LIKE '%Ion%';

NUME	ANI	ANI_CU_LUNI
Ion Ion	34	34.2
Popescu Ion	29	29.1
Ionescu Gheorghe	24	24

10) Să se afişeze numele angajaților şi data împlinirii limitei de vârstă pentru pensionare (62 de ani) precum şi numărul de luni rămase până la pensionare (62ani*12luni).

SQL> SELECT nume,
ADD_MONTHS (datan, 62*12) DATA_PENSIONARE
MONTHS_BETWEEN(ADD_MONTHS(datan,62*12),sysdate)
LUNI_PENSIONARE
FROM DatePers;

NUME	DATA_PENSIONARE	LUNI_PENSIONARE
Ion Ion	10-JAN-32	334.11
Popescu Ion	10-FEB-37	395.11
Ionescu Gheorghe	10-MAR-42	456.11

11) Să afişeze numele angajaților şi ultima zi a lunii corespunzătoare datei de naştere a angajatilor din localitatea Mangalia

SQL> SELECT nume,
LAST_DAY (datan) ULTIMA_ZI_DIN_LUNA
FROM DatePers
WHERE localitate='Mangalia';

NUME	ULTIMA_ZI_DIN_LUNA
Ionescu Gheorghe	31-MAR-80

12) Să se afişeze următoarea zi de Sămbătă (după DataCurentă->sysdate)

SQL> SELECT NEXT_DAY (sysdate,'Saturday') URMATOAREA_SAMBATA FROM DUAL;

```
URMATOAREA_SAMBATA
-----------------------------------
08-OCT-05
```

13) Să se afişeze numele angajaților şi data naşterii acestora într-un format 'MM/YYYY' (M=Month=Luna, Y=Year=An)

SQL> SELECT nume, TO_CHAR(datan,'MM/YYYY') LUNA_AN FROM DatePers;

NUME	LUNA_AN
Ion Ion	01/1970
Popescu Ion	02/1975
Ionescu Gheorghe	03/1980

14) Să se afişeze numele şi CNP-ul angajaților născuți pe 10 ianuarie 1970

SQL> SELECT cnp FROM DatePers WHERE datan= TO_DATE ('10 January 1970', 'dd Month YYYY');

NUME	CNP
Ion Ion	1234567890100

15) Să se afişeze codul angajaților, numele şi salariul acestora indexat cu 5% pentru economişti şi 10% pentru director. Se stabileşte un JOIN pe tabelele DatePers şi DateSal pentru identificarea numelui şi, respectiv, codul angaților al căror salariu va fi indexat. Salariul care nu va fi indexat va fi trecut cu SAL_BAZA în coloana SAL_INDEXAT (opțiunea DEFAULT din funcția DECODE).

SQL> SELECT ds.codang, dp.nume, ds.salbaza SAL_BAZA, DECODE (functia, 'Ec', salbaza*1.05, 'Director', salbaza*1.1, salbaza) SAL_INDEXAT FROM DateSal ds, DatePers dp WHERE ds.codang = dp.codang;

CODANG	NUME	SAL_BAZA	SAL_INDEXAT
100	Ion Ion	5000000	5250000
200	Popescu Ion	6500000	6500000
300	Ionescu Gheorghe	15000000	16500000

16) Să se afişeze suma salariilor de bază

```
SQL>  SELECT 'Suma este' ||sum (salbaza)   SUMA
      FROM DatePers dp, DateSal ds
      WHERE dp.codang=ds.codang(+);
```

```
SUMA
---------------------------
Suma este 26500000
```

17) Să se afişeze numele fiecărui angajat şi codul şefului direct superior

```
SQL>  SELECT nume || 'lucreaza pentru' || codsef  ANGAJAT_SEF
      FROM DatePers dp, DateSal ds
      WHERE dp.codang=ds.codang;
```

```
ANGAJAT_SEF
-----------------------------------------------------------
Ion Ion             lucreaza pentru      300
Popescu Ion         lucreaza pentru      300
Ionescu Gheorghe    lucreaza pentru       -
```

18) Să se afişeze salariu de bază mediu, salariu minim şi salariu maxim pentru toți salariații cu codul cuprins între 10 şi 1000.

```
SQL>  SELECT Avg (salbaza)        MEDIU,
      Min (salbaza)               MINIM,
      Max (salbaza)               MAXIM
      FROM DateSal
      WHERE codang BETWEEN 10 AND 1000;
```

```
MEDIU    MINIM    MAXIM
-----------------------------------------------
 7875000  3500000  16500000
```

19) Să se afişeze toți angajații cu funcția de Director, din localitatea Mangalia şi cu un salariu mai mare de 14000000.

```
SQL>  SELECT nume, functia, salbaza
```

FROM DatePers dp, DateSal ds
WHERE
dp.codang=ds.codang AND
functia= 'Director' AND
dp.localitate='Mangalia' AND
salbaza>14000000;

NUME	FUNCTIA	SALBAZA
Ionescu Gheorghe	Director	16500000

20) Să se afişeze toți angajații din structura ierarhică a societății. Rădăcina arborelui este Directorul .

```
SQL> SELECT LPAD (' ',5*(LEVEL-1)) || codang, functia
     FROM DateSal ds
     START WITH functia='Director'
     CONNECT BY PRIOR codang=codsef;
```

Rezultatul este:

LPAD('',5*(LEVEL-1))||CODANG	FUNCTIA
300	Director
100	Ec
200	Inginer
400	Tehnician

21) Să se blocheze rândurile selectate de o cerere

```
SQL> SELECT * FROM Impozitar
     FOR UPDATE
```

Tabelă blocată pentru update-area tuplurilor:

LINIE	DELA	PANALA	SUMA	PROCENT
1		0	210000	18
2	2100001	5200000	378000	23
3	5200001	8300000	1091000	28
4	8300001	11600000	1959000	34
5	11600001	99999999999	3081000	40

22) Să se adauge un nou angajat în tabela DatePers şi să se selecteze angajatul adăugat după prima literă din nume şi după apartenența sa o localitate.

```
SQL> INSERT INTO DatePers VALUES
     (400, 'Popa Vasile', '1234567890400', '10-APR-1980', 'Zorelelor 12'
     ,'Medgidia', '0721333333');
     SELECT * from DatePers
     WHERE nume LIKE ‘P%’
     AND localitate IN (‘Mangalia’, ‘Medgidia’);
```

CODANG	NUME	CNP	DataN	ADRESA	LOCALITATE	TELEFON
400	Popa Vasile	1234567890400	10-APR-80	Zorelelor12	Medgidia	0721333333

23) Să se adauge datele salariale pentru angajatul nou introdus. Să se selecteze codul, numele şi datele salariale introduse pentru noul angajat.

```
SQL> INSERT INTO DateSal
     VALUES (400,'Tehnician',3500000,4,25,200);

     SELECT ds.codang, dp.nume, ds.functia, ds.salbaza, ds.persintr,
     ds.vechime, ds.codsef
     FROM      DatePers dp, DateSal ds
     WHERE     dp.codang=ds.codang
               AND ds.codang=400
               OR dp.nume = ‘%Vasile’;
```

CODANG	NUME	FUNCTIA	SALBAZA	PERSINTR	VECHIME	CODSEF
400	Popa Vasile	Tehnician	3500000	4	25	200

24) Să se adauge în tabela Pontaj datele pentru noul angajat (cu date introduse de la tastatura).

```
SQL> PROMPT Să se adauge în Tabela Pontaj datele pentru:
     INSERT INTO Pontaj (codang, luna, zilelucr, orezi, zileco, zilecm,
     orelucrate)
     VALUES('&CodAngajat',’&LunaPontaj','&ZileLucr','&OrePeZi',
     '&ZileConOdihna','&ZileConMed',' &OreLucrEfectiv');
```

```
Să se adauge în Tabela Pontaj datele pentru:
Enter value for codangajat:        400
Enter value for lunapontaj:        1
Enter value for zilelucr:          22
Enter value for orepezi:           8
Enter value for zileconodihna:  1
Enter value for zileconmed:        1
Enter value for orelucrefectiv:  8
1 row created.
```

Ulterior de poate adăuga la linia de stare (o selecție explicită, prin introducerea codului corespunzător noul angajat inserat în tabelă) .

SQL> SELECT * FROM Pontaj
WHERE codang=&CodAngajat;

Enter value for codangajat: 400

CODANG	LUNA	ZILELUCR	OREZI	ZILECO	ZILECM	ORELUCRATE
400	1	22	8	1	1	8

SAU, o selecție implicită prin specificarea directă a codului angajatului:

SQL> SELECT * FROM Pontaj
WHERE codang= 400;

CODANG	LUNA	ZILELUCR	OREZI	ZILECO	ZILECM	ORELUCRATE
400	1	22	8	1	1	8

25) Să se adauge o noua taxă, în tabela TAXE, utilizând variabile de memorie

```
SQL> ACCEPT den        PROMPT    'Denumire:'
     ACCEPT procent    PROMPT    'Procent:'
     ACCEPT cotamax    PROMPT    'Cota maxima:'
     INSERT INTO Taxe VALUES('&den','&procent','&cotamax');
```

```
Denumire:        TAXA NOUA
Procent:                 2
Cota maxima:             3

Old    1: INSERT INTO Taxe VALUES    ('&den','&procent','&cotamax')
New    1: INSERT INTO Taxe VALUES    ('TAXA NOUA','2','3')
1 row created.
```

Ulterior, după rulare, se va putea selecta.

SQL> SELECT * FROM TAXE
WHERE den = '&Denumire ';

DEN	PROCENT	COTAMAX
TAXA NOUA	2	3

26) Să se creeze o nouă tabelă pentru Datele Personale ale Angajatilor din Constanța (DatePersCta) şi să se adauge ulterior în această tabelă datele personale ale angajaților din Constanța existente în tabela inițială DatePers.

```
SQL> CREATE TABLE DatePersCta
     (
     codang number(5) primary key,
     nume varchar2(35),
     cnp varchar2(13),
     datan date,
     adresa varchar2(30),
     localitate varchar2(15),
     telefon varchar2(10)
     );

     INSERT INTO DatePersCta
     SELECT * FROM DatePers
     WHERE localitate='Constanta';

     COMMIT;

SELECT * FROM DatePersCta;
```

CODANG	NUME	CNP	DataN	ADRESA	LOCALITATE	TELEFON
100	Ion Ion	1234567890100	10-JAN-1970	Mangaliei 100	Constanta	0722123456
200	Popescu Ion	1234567890200	10-FEB-1975	Tomis 232	Constanta	0744123456

27) Să se majoreze salariul directorului cu 10 procente.

```
SQL> UPDATE DateSal
     SET salbaza=salbaza*1.1
     WHERE functia='Director';
```

Rezultatul se poate vizualiza utilizând variabila "Functia":

```
SQL> SELECT * FROM DateSal
     WHERE functia='&Functia';
```

Enter value for functia: Director

NRCRT	CODANG	FUNCTIA	SALBAZA	PERSINTR	VECHIME	CODSEF
3	300	Director	19965000	0	15	

28) Să se şteargă toate înregistrările din DatePersCta unde numărul de telefon începe cu “0744...”

```
SQL>  DELETE FROM DatePersCta
      WHERE telefon LIKE '0744%';

      SELECT * FROM DatePersCta;
```

CODANG	NUME	CNP	DataN	ADRESA	LOCALITATE	TELEFON
100	Ion Ion	1234567890100	10-JAN-1970	Mangaliei 100	Constanta	0722123456

29) Să se afişeze numele tabelelor create în schema proprie de obiecte

```
SQL>  SELECT table_name from USER_TABLES;
```

```
TABLE_NAME
-------------------
DATEPERS
DATEPERSCTA
DATESAL
DEDUCERI
DEPT
EMP
IMPOZITAR
PONTAJ
SALGRADE
TAXE
```

30) Să se adauge atributul TMP de tip NUMBER în tabela DatePersCta.

```
SQL>  ALTER TABLE DatePersCta
      ADD ( TMP NUMBER  (3) );

      DESCRIBE DatePersCta;
```

Name	Null?	Type
CODANG	NOT NULL	NUMBER(5)
NUME		VARCHAR2(35)
CNP		VARCHAR2(13)
DATAN		DATE
ADRESA		VARCHAR2(30)
LOCALITATE		VARCHAR2(15)
TELEFON		VARCHAR2(10)
TMP		NUMBER (3)

31) Să se modifice atributul TMP la o lungime de 5 poziții

```
SQL>  ALTER TABLE DatePersCta
```

MODIFY (TMP NUMBER (5));

DESCRIBE DatePersCta;

```
Name                Null?               Type
------------------------------------------------------------------------
CODANG              NOT NULL            NUMBER(5)
NUME                                    VARCHAR2(35)
CNP                                     VARCHAR2(13)
DATAN                                   DATE
ADRESA                                  VARCHAR2(30)
LOCALITATE                              VARCHAR2(15)
TELEFON                                 VARCHAR2(10)
TMP                                     NUMBER (5)
```

32) Să se redenumească tabela DatePersCta în CONST

SQL> ALTER TABLE DatePersCta
RENAME TO Const;

33) Să se şteargă tabela DatePersCta

SQL> DROP TABLE DatePersCta;

34) Să se adauge la DatePers restricția de Validare Codang>0.

SQL> ALTER Table DatePers
ADD (CONSTRAINT check_comp CHECK (codang>0));

35) Să se creeze tabela virtuală CONSTANTA care va conține date despre angajații din Constanța

SQL> CREATE VIEW Constanta
AS SELECT *
FROM DatePers
WHERE localitate='Constanta';

View created.

36) Să se şteargă tabela virtuală CONSTANTA

SQL> DROP VIEW Constanta;

View dropped.

37) Să se afişeze numărul de înregistrări din tabela DatePers

```
SQL>  SELECT count (*) NR_INREG
      FROM DatePers;
```

```
NR_INREG
---------------
         4
```

38) Să se vizualizeze restricțiile tabelei DatePers

```
SQL>  SELECT
          CONSTRAINT_TYPE,
          CONSTRAINT_NAME,
          STATUS
      FROM USER_CONSTRAINTS
      WHERE TABLE_NAME= 'DATEPERS';
```

```
C CONSTRAINT_NAME        STATUS
- ------------------------------ -----------------------
C CHECK_COMP             ENABLED
```

7.2. Aplicație informatică pentru activitatea de aprovizionare şi desfacere a unei firme

1. Crearea bazei de date.

1) Să se creeze tabelele clienți, furnizori, produse, tranzacții, documente şi proddoc.

```
CREATE TABLE clienti
(
    codc        varchar2 (5),
    denc        varchar2 (30),
    adr         varchar2 (30),
    loc         varchar2 (20),
    cont        varchar2 (11),
    banca       varchar2 (15),
        constraint pk_codc primary key (codc)
);

CREATE TABLE furnizori
(
    codf        varchar2 (5),
    denf        varchar2 (30),
    adr         varchar2 (30),
    loc         varchar2 (20),
    cont        varchar2 (11),
    banca       varchar2 (15),
        constraint pk_codf primary key (codf)
);

CREATE TABLE produse
(
    codp        varchar2 (5),
    denp        varchar2 (25),
    um          varchar2 (5),
    pret        number (10),
    stoc        number (5),
    termen      date,
        constraint pk_codp primary key (codp)
);
```

```
CREATE TABLE tranzactii
(
    codt        varchar2 (5),
    dent        varchar2 (1)
        constraint nn_dent not null
        constraint ck_dent check (upper (dent) in ('L','R')),
    dataora     date default sysdate,
    codf        varchar2 (5),
    codc        varchar2 (5),
        constraint pk_codt primary key (codt),
        constraint fk_codf foreign key (codf) references furnizori (codf),
        constraint fk_codc foreign key (codc) references clienti (codc)
    );

CREATE TABLE documente
(
    codd        number (5)
        constraint ck_codd check (codd>0),
    dend        varchar2 (4)
        constraint nn_dend not null
        constraint ck_dend check (upper (dend) in
    ('FACT','AVIZ','NIR','CHIT')),
    data        date default sysdate,
    codt        varchar2 (5),
        constraint pk_codd primary key (codd),
        constraint fk_codt foreign key (codt) references tranzactii (codt)
    );

CREATE TABLE proddoc
 (
    codd        number(5),
    codp        varchar2(5),
    um          varchar2(5),
    cant         number(5),
        constraint pk_coddp primary key (codd,codp)
);
```

2. Modificarea structurii tabelelor bazei de date

1) Să se modifice dimensiunea atributului CodP din tabela Produse, la 4 caractere.

SQL> PROMPT Modificati dimensiunea atributului Codp din tabela Produse la 4 caractere
SQL> ALTER TABLE produse MODIFY (codp varchar2 (4));
SQL> DESCRIBE produse;

Name	Null?	Type
CODP	NOT NULL	VARCHAR2 (4)
DENP		VARCHAR2 (40)
UM		VARCHAR2 (5)
PRET		NUMBER (13)
STOC		NUMBER (7)
TERMEN		DATE

2) Adăugați atributul IE (number(2)) tabelei ProdDoc

SQL> PROMPT Adaugati atributul IE (number (2)) tabelei proddoc
SQL> ALTER TABLE proddoc ADD (IE NUMBER (2));
SQL> DESCRIBE proddoc;

Name	Null?	Type
CODD	NOT NULL	NUMBER (5)
CODP	NOT NULL	VARCHAR2 (5)
UM		VARCHAR2 (5)
CANT		NUMBER (6)
IE		NUMBER (2)

3) Adăugați atributul Valoare, numeric de 20 caractere, la tabela Documente.

SQL> PROMPT Adaugati atributul valoare la tabela documente
SQL> ALTER TABLE documente ADD (valoare number (20));
SQL> DESCRIBE documente;

Name	Null?	Type
CODD	NOT NULL	NUMBER (5)
DEND	NOT NULL	VARCHAR2 (4)
DATA		DATE
CODT		VARCHAR2 (5)
VALOARE		NUMBER (20)

3.Inserare înregistrări în tabele.
SQL> DELETE FROM clienti;
SQL> DELETE FROM furnizori;

```
SQL>  DELETE FROM produse;
SQL>  DELETE FROM tranzactii;
SQL>  DELETE FROM documente;
SQL>  DELETE FROM proddoc;

SQL>  PROMPT INSERARE ÎN TABELA CLIENȚI;
SQL>  INSERT INTO clienti VALUES ('1','GOODS ','PIPERA
      135','BUCURESTI','A1234567890','BRD');
SQL>  INSERT INTO clienti VALUES ('2','DepozitPC','Stefan cel
      Mare 110','Bucuresti', 'A1231231234','BCR');
SQL>  INSERT INTO clienti VALUES ('3','Flamingo','Mihai
      Eminescu 18','Cluj','A1231231235','BCR');
SQL>  INSERT INTO clienti VALUES ('4','Ultra Pro','Mihai Bravu
      11','Timisoara','B1231231234','BRD');
SQL>  INSERT INTO clienti VALUES ('5','Flanco','Dorobantilor
      130','Cluj','C1231231234','BCR');

SQL>  PROMPT INSERARE ÎN TABELA FURNIZORI;
SQL>  INSERT INTO furnizori VALUES  ('1','GOODS ','PIPERA
      135','BUCURESTI','A1234567890','BRD');
SQL>  INSERT INTO furnizori VALUES  ('2','ComputerNT','Gral
      Popescu 13','Iasi','A1234123412','BRD');
SQL>  INSERT INTO furnizori VALUES  ('3','Python','Charles de
      Gaule 117','Cluj','A1234512345','BCR');
SQL>  INSERT INTO furnizori VALUES  ('4','Blue Ridge','Magheru
      307','Bucuresti','B1234554321','BRD');
SQL>  INSERT INTO furnizori VALUES  ('5','Deck
      Electronics','Lacul Alb 35','Iasi','B1234567777','BCR');

SQL>  PROMPT INSERARE ÎN TABELA PRODUSE;
SQL>  INSERT INTO produse VALUES('P1','Monitor
      7inch','buc',3500000,1000,
      TO_DATE('01/08/2006','DD/MM/YYYY'));
SQL>  INSERT INTO produse VALUES('P2','CD-RW ASUS
      24x10x40x','buc',1000000,500,
      TO_DATE('01/08/2005','DD/MM/YYYY'));
SQL>  INSERT INTO produse VALUES('P3','Tastatura
      qwerty','buc',300000,100,
      TO_DATE('01/06/2004','DD/MM/YYYY'));
```

```
SQL>  INSERT INTO produse VALUES('P4','CPU AMD Athlon
      1.4GHz','buc',2700000,700,
      TO_DATE('01/12/2004','DD/MM/YYYY'));
SQL>  INSERT INTO produse VALUES('P5','Mouse
      A4TECH','buc',100000,150,
      TO_DATE('01/06/2004','DD/MM/YYYY'));

SQL>  PROMPT INSERARE ÎN TABELA TRANZACȚII;
SQL>  INSERT INTO tranzactii VALUES
      ('T1','R',TO_DATE('01/08/2003 02:12:39','MM/DD/YYYY
      HH:MI:SS'),'3','1');
SQL>  INSERT INTO tranzactii VALUES
      ('T2','R',TO_DATE('11/10/2003 10:20:09','MM/DD/YYYY
      HH:MI:SS'),'4','1');
SQL>  INSERT INTO tranzactii VALUES
      ('T3','L',TO_DATE('12/10/2003 12:12:30','MM/DD/YYYY
      HH:MI:SS'),'1','5');
SQL>  INSERT INTO tranzactii VALUES
      ('T4','L',TO_DATE('02/11/2003 04:55:39','MM/DD/YYYY
      HH:MI:SS'),'1','2');

SQL>  PROMPT INSERARE ÎN TABELA DOCUMENTE;
SQL>  INSERT INTO documente (codd,dend,data,codt) VALUES
      (10123,'FACT',TO_DATE('01/08/2003','MM/DD/YYYY'),'T1');
SQL>  INSERT INTO documente (codd,dend,data,codt) VALUES
      (20123,'NIR',TO_DATE('01/08/2003','MM/DD/YYYY'),'T1');
SQL>  INSERT INTO documente (codd,dend,data,codt) VALUES
      (10124,'FACT',TO_DATE('11/10/2003','MM/DD/YYYY'),'T2');
SQL>  INSERT INTO documente (codd,dend,data,codt) VALUES
      (20124,'NIR',TO_DATE('11/10/2003','MM/DD/YYYY'),'T2');
SQL>  INSERT INTO documente (codd,dend,data,codt) VALUES
      (30122,'AVIZ',TO_DATE('12/10/2003','MM/DD/YYYY'),'T3');
SQL>  INSERT INTO documente (codd,dend,data,codt) VALUES
      (10125,'FACT',TO_DATE('12/10/2003','MM/DD/YYYY'),'T3');
SQL>  INSERT INTO documente (codd,dend,data,codt) VALUES
      (30123,'AVIZ',TO_DATE('02/11/2003','MM/DD/YYYY'),'T4');
SQL>  INSERT INTO documente (codd,dend,data,codt) VALUES
      (10126,'FACT',TO_DATE('02/11/2003','MM/DD/YYYY'),'T4');
SQL>  INSERT INTO documente (codd,dend,data,codt) VALUES
      (40123,'CHIT',TO_DATE('02/11/2003','MM/DD/YYYY'),'T4');
```

Valorile pentru câmpul valoare din tabela Documente nu au fost direct introduse în tabelă, deoarece acest câmp este unul calculat, iar valorile sale se vor introduce printr-o formulă.

SQL> PROMPT INSERARE ÎN TABELA PRODDOC;
SQL> INSERT INTO proddoc VALUES (10123,'P1','buc',500,null);
SQL> INSERT INTO proddoc VALUES (10123,'P2','buc',500,null);
SQL> INSERT INTO proddoc VALUES (20123,'P1','buc',500,null);
SQL> INSERT INTO proddoc VALUES (20123,'P2','buc',500,null);
SQL> INSERT INTO proddoc VALUES (10124,'P3','buc',100,null);
SQL> INSERT INTO proddoc VALUES (10124,'P4','buc',500,null);
SQL> INSERT INTO proddoc VALUES (10124,'P5','buc',100,null);
SQL> INSERT INTO proddoc VALUES (20124,'P3','buc',100,null);
SQL> INSERT INTO proddoc VALUES (20124,'P4','buc',450,null);
SQL> INSERT INTO proddoc VALUES (20124,'P5','buc',100,null);
SQL> INSERT INTO proddoc VALUES (30122,'P1','buc',100,null);
SQL> INSERT INTO proddoc VALUES (30122,'P2','buc',200,null);
SQL> INSERT INTO proddoc VALUES (10125,'P1','buc',100,null);
SQL> INSERT INTO proddoc VALUES (10125,'P2','buc',200,null);
SQL> INSERT INTO proddoc VALUES (30123,'P1','buc',300,null);
SQL> INSERT INTO proddoc VALUES (30123,'P4','buc',500,null);
SQL> INSERT INTO proddoc VALUES (10126,'P1','buc',300,null);
SQL> INSERT INTO proddoc VALUES (10126,'P4','buc',500,null);

4. Definirea generatorului de numere de secvență:
1) Să se creeze o secvență SECV care începe cu valoarea 10127 şi se termina cu valoarea 10130 şi pasul 1. Acestă secvență secv se va folosi ulterior pentru generarea automată de numere unice pentru câmpul codd din tabela Documente. Se vor genera succesiv, crescător, numerele cuprinse între 10127 şi 10130.

```
SQL>CREATE SEQUENCE secv     // nume secvență
        INCREMENT BY 1       // pasul de incrementare
        START WITH 10127     // valoarea de pornire a secvenței
        MAXVALUE 10130       // valoarea maximă a secvenței
        NOCACHE NOCYCLE;     // secvență finită
```

2) Să se adauge o nouă valoare pentru atributul cheii primare codd din tabela Documente folosindu-se succesiunea generată de secvența SECV anterior creată.

SQL> INSERT INTO documente VALUES (secv.nextval, 'FACT', sysdate-2, 'T5', null);
SQL> SELECT * from documente;

CODD	DEND	DATA	CODT	VALOARE
10123	FACT	08-JAN-05	T1	
20123	NIR	08-JAN-05	T1	
10124	FACT	10-NOV-05	T2	
20124	NIR	10-NOV-05	T2	
30122	AVIZ	10-DEC-05	T3	
10125	FACT	10-DEC-05	T3	
30123	AVIZ	11-FEB-05	T4	
10126	FACT	11-FEB-05	T4	
40123	CHIT	11-FEB-05	T4	
10127	FACT	26-FEB-06	T5	

La execuție se observă adăugarea tuplului 10127-FACT-26FEB04-T5, cheia primară astfel definită, pentru câmpul codd, fiind prima valoare a secvenței SECV. La fiecare apelare a cuplului INSERT-SELECT secvența SECV anterior creată va incrementa automat cheia primară Codd din tabela Documente.

3) Să se adauge înregistrările corespunzătoare pentru o recepția a 100 de bucăți din produsul P3 şi alte 200 de bucăți din produsul P4 de la furnizorul 4, ştiindu-se că factura a fost emisă de furnizor cu 2 zile înainte de recepția produselor.

SQL> INSERT INTO tranzactii VALUES ('T5','R', sysdate, '4','1');
SQL> INSERT INTO documente VALUES (secv.nextval, 'FACT', sysdate-2, 'T5',0);
SQL> INSERT INTO documente VALUES (20125,'NIR', sysdate,'T5' ,0);
SQL> INSERT INTO proddoc VALUES (secv.currval,'P3','buc',100,0);*
SQL> INSERT INTO proddoc VALUES (secv.currval,'P4','buc',200,0);*
SQL> INSERT INTO proddoc VALUES (20125,'P3','buc',100,1);
SQL> INSERT INTO proddoc VALUES (20125,'P4','buc',200,1);

Pentru a se putea defini cerința ca factura să fie emisă cu două zile înainte de recepția produselor, s-a optat pentru varianta sysdate-2 (data curentă-două zile), întrucât recepția produselor intrate în gestiune se face în ziua curentă de lucru. Cele două tupluri sunt cele care definesc aprovizionarea produselor P3 şi P4, având drept valori, pentru unul din cele două atribute ale cheii primare, codd - numărul de secvență curent (secv.currval) definit anterior şi preluat de la generatorul secv.nextval (din înregistrarea: secv.nextval - FACT- sysdate-2, T5, 0)

Adăugările la stoc ale celor două produse se vor regăsi în tabela Proddoc.

Înainte de inserare:

SQL> select * from documente;

CODD	DEND	DATA	CODT
20123	**NIR**	**08-JAN-05**	**T1**
10124	**FACT**	**10-NOV-05**	**T2**
20124	**NIR**	**10-NOV-05**	**T2**
30122	**AVIZ**	**10-DEC-05**	**T3**
10125	**FACT**	**10-DEC-05**	**T3**
30123	**AVIZ**	**11-FEB-05**	**T4**
10126	**FACT**	**11-FEB-05**	**T4**
40123	**CHIT**	**11-FEB-05**	**T4**
10123	**FACT**	**08-JAN-05**	**T1**

După inserare:

SQL> select * from documente;

CODD	DEND	DATA	CODT	VAL
20123	**NIR**	**08-JAN-05**	**T1**	
10124	**FACT**	**10-NOV-05**	**T2**	
20124	**NIR**	**10-NOV-05**	**T2**	
30122	**AVIZ**	**10-DEC-05**	**T3**	
10125	**FACT**	**10-DEC-05**	**T3**	
30123	**AVIZ**	**11-FEB-05**	**T4**	
10126	**FACT**	**11-FEB-05**	**T4**	
40123	**CHIT**	**11-FEB-05**	**T4**	
10123	**FACT**	**08-JAN-05**	**T1**	
10127	**FACT**	**27-FEB-06**	**T5**	**0**
20125	**NIR**	**29-FEB-06**	**T5**	**0**

Câmpurile adăugate în cele două tabele şi având valoarea cheii primare generată ca fiind 10127 (primul număr din secvența SECV) sunt cele corespunzătoare instrucțiunilor SECV.NEXTVAL şi SECV.CURRVAL, care au generat, respectiv, preluat valorile pentru cheia primară.

4) Să se afişeze ultimul număr utilizat din secvenţa SECV:

SQL> SELECT secv.CURRVAL
FROM DUAL;

CURRVAL

10127

5) Să se modifice pasul secvenţei SECV de la 1 la 10000

SQL> ALTER SEQUENCE secv
INCREMENT BY 10000;
Sequence altered.

6) Să se şteargă secvenţa SECV .

SQL> DROP SEQUENCE secv;
Sequence dropped.

5. Actualizări la nivelul aplicaţiei:

1) Să se insereze în atributul IE din tabela Proddoc, valorile: “1”,pentru NIR (I) ; „-1”, pentru AVIZE (E); “0”, pentru celelalte documente.

```
SQL> UPDATE proddoc SET IE= -1
     WHERE SUBSTR (TO_CHAR (codd), 1, 1)= ' 3 ';
SQL> UPDATE proddoc SET IE=1
     WHERE SUBSTR (TO_CHAR (codd), 1, 1)= ' 2 ';
SQL> UPDATE proddoc SET IE=0
     WHERE SUBSTR (TO_CHAR (codd), 1, 1) NOT IN( '2 ', ' 3 ');
SQL> SELECT * FROM proddoc;
```

CODD	CODP	UM	CANT	IE
10123	P2	buc	500	0
20123	P1	buc	500	1
10123	P1	buc	500	0
20123	P2	buc	500	1
30123	P1	buc	300	-1
30123	P4	buc	500	-1
10126	P1	buc	300	0
10126	P4	buc	500	0
10127	P3	buc	100	0
10127	P4	buc	200	0
20125	P3	buc	100	1
20125	P4	buc	200	1

2) Să se calculeze şi să se afişeze valoarea totală pentru fiecare document, din tabela Documente.

```
SQL>  UPDATE documente D SET
      valoare =
      (SELECT SUM (cant*pret) valoare
      FROM proddoc PD, produse P
      WHERE  P.codp=PD.codp AND D.codd=PD.codd
      GROUP BY D.codd);
SQL>  SELECT codd, valoare
       FROM documente
      WHERE valoare IS NOT NULL;
```

CODD	VALOARE
20123	2.250E+09
10124	1.390E+09
20124	1.255E+09
30122	550000000
10125	550000000
30123	2.400E+09
10126	2.400E+09
10127	570000000
20125	570000000
10123	2.250E+09

3) Să se diminueze stocul aferent prodului P5 cu 50 de bucăți.

```
SQL>  UPDATE produse
            SET stoc= stoc - 50
            WHERE codp= 'P5';
SQL>  SELECT  codp, denp, stoc  from produse ;
```

CODP	DENP	STOC
P5	Mouse A4TECH	150

6. Funcțiile pentru şiruri de caractere:

1) Să se selecteze numele şi localitea unde îşi au sediul clienții, folosind formatul de afişare cu prima literă majusculă.

```
SQL>  SELECT
      INITCAP (DENC) LITERA_MARE_NUME,
      INITCAP (LOC)
```

FROM clienti;

LITERA_MARE_NUME	INITCAP (LOC)
Interconn	Bucuresti
Depozitul De Calculatoare	Bucuresti
Flamingo	Cluj
Ultra Pro	Timisoara
Flanco	Cluj

2) Să se concateneze şirurile corespunzătoare atributelor „Adresa" şi „Localitate" din tabela furnizori, pentru furnizorul "Blue Ridge" .

```
SQL> SELECT denf, CONCAT (adr, loc)  "Adesa_din_Localitatea"
     FROM Furnizori
     WHERE denf = 'Blue Ridge ';
```

DENF	Adresa _ din _ Localitatea
Blue Ridge	Magheru 307 Bucuresti

3) Să se selecteze toţi furnizorii, aducând coddf, denf şi loc la lungimea de 20 de caractere fiecare, utilizând LPAD şi RPAD.

```
SQL> SELECT    LPAD (codf, 20, ' * '),
               LPAD (denf, 20),
               LPAD  (loc, 20, '-' )
     FROM furnizori ;
```

LPAD(CODF,20,'*')	LPAD(DENF,20)	LPAD(LOC,20,'-')
*******************1	INTERCONN	-----------Bucureşti
*******************2	Computer Network	------------------Iasi
*******************3	Python	------------------Cluj
*******************4	Blue Ridge	-----------Bucuresti
*******************5	Deck Electronics	-------------------Iasi

```
SQL> SELECT    RPAD (codf, 20, ' * '),
               RPAD (denf, 20),
               RPAD  (loc, 20, '-' )
     FROM furnizori ;
```

RPAD(CODF,20,'')*	*RPAD(DENF,20)*	*RPAD(LOC,20,'-')*
*1 * * * * * **	*INTERCONN*	*BUCURESTI----*
*2 * * * * * **	*Computer Network*	*Iasi-----------------*
*3 * * * * * **	*Python*	*Cluj----------------*
*4 * * * * * **	*Blue Ridge*	*Bucuresti----------*
*5 * * * * * **	*Deck Electronics*	*Iasi-----------------*

4) Să se afişeze furnizorii din alte localităţi decât Bucureşti.

```
SQL>  SELECT codf, denf , loc FROM furnizori
      WHERE UPPER (loc) <> 'BUCURESTI' ;
```

CODF	DENF	LOC
2	Computer Network	Iasi
3	Python	Cluj
5	Deck Electronics	Iasi

5) Să se afişeze clienții a căror denumire începe cu litera "F"

```
SQL>  SELECT codc, denc FROM clienti
      WHERE SUBSTR (denc,1,1)= 'F';
```

CODC	DENC
3	Flamingo
5	Flanco

9. Funcțiile de dată

1) Să se afişeze denumirea furnizorilor cu care nu s-au mai încheiat tranzacţii în ultimele 6 luni.

```
SQL>  SELECT codf, denf  FROM furnizori
      WHERE codf NOT IN
            (
            SELECT codf FROM tranzactii
            WHERE MONTHS_BETWEEN (sysdate,dataora)<=6
            );
```

CODF	DENF
2	Computer Network
3	Python
5	Deck Electronics

2) Să se afişeze perioada (lunile) de garanţie rămase până la expirarea produselor (inventariate în tabela Produse) cu enumerarea doar a celor care mai au ca valabilitate minimum 3 luni.

```
SQL>  SELECT codp, denp,
```

```
         MONTHS_BETWEEN (termen, sysdate) LUNI_GARANTIE
         FROM produse
         where MONTHS_BETWEEN (termen, sysdate) >3;
```

CODP	DENP	LUNI_GARANTIE
P1	Monitor 17inch	29.0727
P2	CD-RW ASUS 24x10x40x	17.0727
P3	Tastatura qwerty	3.0727001
P4	CPU AMD Athlon 1.4GHz	9.0727001
P5	Mouse A4TECH	3.0727001

3) Să se selecteze produsul cu termenul de garanție cel mai îndepărtat (August 2007) şi să se evidențieze lunile de garanție rămase de la data curentă la termen.

```
SQL> SELECT codp, denp,
         MONTHS_BETWEEN  ('01-Aug-06', sysdate)
         LUNI_MAXIME_GARANTIE
         FROM produse
         WHERE termen='01-Aug-07' ;
```

CODP	DENP	LUNI_MAXIME_GARANTIE
P1	Monitor 17 inch	29.072489

4) Să se afişeze, codul, denumirea, termenul de garanție, precum şi data decalată cu trei luni față de termenul de garanție şi data anetrioară cu trei luni termenului de garanție. Se vor evidenția produsele ale căror termene de valabilitate nu au expirat.

```
SQL> SELECT codp, denp, termen,
         ADD_MONTHS (termen, 3)  PESTE_TREI_L,
         ADD_MONTHS (termen, -3) CU_TREI_L_IN_URMA
         FROM produse
         WHERE termen>sysdate;
```

CODP	DENP	TERMEN	PESTE_TREI_L	CU_TREI_L_ÎN_URMĂ
P1	Monitor 17inch	01-AUG-07	01-NOV-07	01-MAY-07
P2	CD-RW AS	01-AUG-06	01-NOV-06	01-MAY-06
P3	Tastatura	01- JUN-05	01- SEP- 05	01-MAR-05
P4	CPU AMD 1.4GHz	01- DEC-05	01-MAR-06	01-SEP- 05
P5	Mouse A4TECH	01- JUN-05	01- SEP- 05	01-MAR-05

5) Să se afişeze data următoarei zile a săptămânii *(char)* după o dată declarată.

SQL> SELECT NEXT_DAY ('01-MAR-05', 1)
FROM dual;

NEXT_DAY

06-MAR-05

SQL> SELECT NEXT_DAY ('01-MAR-05', 2)
FROM dual;

NEXT_DAY

07-MAR-05

5) Să se afişeze ultima zi a lunii *(char)* după o dată declarată.

SQL> SELECT LAST_DAY ('01-jun-05')
FROM dual;

LAST_DAY

30-JUN-05

SQL> SELECT codp, denp, termen,
LAST_DAY (termen) ULTIMA_ZI_LUNA
FROM produse;

CODP	DENP	TERMEN	ULTIMA_ZI_LUNA
P1	Monitor 17inch	01-AUG-07	31- AUG-07
P2	CD-RW ASUS 24x10x40x	01-AUG-06	31 -AUG-06
P3	Tastatura qwerty	01 -JUN-05	30 - JUN-05
P4	CPU AMD Athlon 1.4GHz	01 -DEC-05	31 - DEC-05
P5	Mouse A4TECH	01 -JUN-05	30 -JUN-05.

Funcția *ROUND* poate fi aplicată pe date calendaristice. *Round (data1)*întoarce *data1* cu timpul setat la 12:00AM (noaptea). Aceasta este folositoare atunci când se compară date care au timpuri diferite.

ROUND (data1,'MONTH') întoarce:

- prima zi a lunii conținând *data1*, dacă *data1* este în prima parte a lunii,
- prima zi a următoarei luni, dacă *data1* este în a doua jumătate a lunii

- *ROUND(data 1,' YEAR')* întoarce:
- prima zi a anului conținând *data1*, dacă *data1* este în prima jumătate a anului,
- prima zi a urmatorului an, dacă *data1* este în a doua jumătate a lunii

De exemplu:
6) Să se folosescă funcția ROUND pentru a returna prima zi a lunii sau anului sau prima zi a următoarei luni sau an, în funcție de data declarată.

```
SQL>  SELECT SYSDATE,
      ROUND (SYSDATE, 'MONTH' ) LUNA_ROTUNJITA,
      ROUND (SYSDATE,'YEAR') ANUL_ROTUNJIT
      FROM DUAL;
```

SYSDATE	LUNA_ ROTUNJITA	ANUL_ ROTUNJIT
02-SEP-05	01-SEP-05	01-JAN-06

7) Analog, să se identifice rezultatele întoarse de funcția LAST_DAY pentru valorile atributului TERMEN din tabela Produse:

```
SQL>  SELECT codp, denp, termen,
      LAST_DAY (termen) ULTIMA_ZI_LUNA
      FROM produse;
```

CODP	DENP	TERMEN	ULTIMA_ZI_LUNA
P1	Monitor 17inch	01-AUG-07	31-AUG-07
P2	CD-RW ASUS 24x10x40x	01-AUG-06	31-AUG-06
P3	Tastatura qwerty	01-JUN-05	30-JUN-05
P4	CPU AMD Athlon 1.4GHz	01-DEC-05	31-DEC-05
P5	Mouse A4TECH	01-JUN-05	30-JUN-05

8) Funcția ***TRUNC(data1,'char')*** găseşte prima zi a lunii care e conținută în *data1*, dacă *char = 'MONTH' sau* găseşte prima zi a anului care conține *data1*dacă *char= 'YEAR'*. Să se utilizeze facilitățile acestei funcții.

```
SQL>  SELECT SYSDATE DATA_CURENTA,
      TRUNC (SYSDATE, 'MONTH') PRIMA_ZI_LUNA,
      TRUNC (SYSDATE,'YEAR') PRIMA_ZI_AN
       FROM  SYS.DUAL;
```

```
DATA_CURENTA    PRIMA_ZI_LUNA  PRIMA_ZI_AN
--------- --------- ------------------------------------------------------
02-OCT-05              01-OCT-05             01-JAN-05
```

8. Funcții matematice:

1) Să se afişeze lungimea atributului Denumire client din tabela Clienți

SQL> SELECT denc,
LENGTH (denc) LUNGIME_NUME
FROM clienti;

```
DENC                        LUNGIME_NUME
------------------------------ -----------------------
INTERCONN                        9
Depozitul de calculatoare       25
Flamingo                         8
Ultra Pro                        9
Flanco                           6
```

2) Să se afişeze comisionul corespunzător vânzării fiecărui produs, în mii lei

SQL> ACCEPT comision PROMPT 'Introduceti comision: ';
SQL> SELECT denp, pret,
& comision COMISION (%)
pret*&comision/1000 VALOARE_COMISION
FROM produse;

```
Introduceti comision:            10

DENP                              PRET     COMISION (%) VALOARE_COMISION
---------------------------------------- ------------------------------------------------------------
Monitor 17inch                  3500000         10                           35000
CD-RW ASUS 24x10x40x            1000000         10                           10000
Tastatura qwerty                 300000         10                            3000
CPU AMD Athlon 1.4GHz           2700000         10                           27000
Mouse A4TECH                     100000         10                            1000
```

3) Să se calculze şi afişeze stocul inițial pentru fiecare produs în parte.

SQL> SELECT P.codp,
stoc STOC_INITIAL,
SUM (stoc+IE*cant) STOC_CURENT
FROM produse P, proddoc PD

```
WHERE       P.codp = PD.codp
GROUP BY    P.codp, stoc;
```

CODP	STOC_INITIAL	STOC_CURENT
P1	1000	6100
P2	500	2300
P3	100	600
P4	700	4350
P5	100	300

4) Să se afişeze denumirea, preţul şi stocul actual al produselor, sub forma: PRODUSUL <<nume>> ARE PRETUL UNITAR: <<pret>> LEI. STOCUL ACTUAL ESTE: <<stoc>> <<um>>

```
SQL> SELECT 'PRODUSUL ' || LOWER (denp)
     ||  'ARE PRETUL UNITAR: '  ||pret||
     'LEI. STOCUL ACTUAL ESTE: ' ||stoc||
     'DE' || um
     FROM produse;
```

PRODUSUL Monitor 17inch ARE PRETUL UNITAR: 3500000 LEI. STOCUL ACTUAL ESTE: 1000 DE buc
PRODUSUL Cd-rw asus 24x10x40x ARE PRETUL UNITAR: 1000000 LEI. STOCUL ACTUAL ESTE: 500 DE buc
PRODUSUL Tastatura qwerty ARE PRETUL UNITAR: 300000 LEI. STOCUL ACTUAL ESTE: 100 DE buc
PRODUSUL CPU amd athlon 1.4ghz ARE PRETUL UNITAR: 2700000 LEI. STOCUL ACTUAL ESTE: 700 DE buc
PRODUSUL Mouse a4tech ARE PRETUL UNITAR: 100000 LEI. STOCUL ACTUAL ESTE: 100 DE buc

5) Să se afişeze codul produsului şi preţul mărit cu 1.1 pentru Monitoare şi cu 1.2 pentru Mouse.

```
SQL> SELECT codp,
     pret  PRET_INITIAL,
     DECODE      (denp, 'monitor 17inch', pret*1.1,
                 ‘mouse A4TECH', pret*1.2,  pret) PRET_MARIT
     FROM produse;
```

CODP	PRET_INITIAL	PRET_MARIT
P1	3500000	3850000
P2	1000000	1000000
P3	300000	300000
P4	2700000	2700000
P5	100000	120000

6) Să se afieze Cantitatea Medie cumpărată din fiecare produs şi să se ordoneze tuplurile după Cantitate.

```
SQL> SELECT P.codp, AVG (cant) CANT_MEDIE
     FROM produse P, proddoc PR
     WHERE P.codp= PR.codp AND IE =1
     GROUP BY P.codp
     ORDER BY AVG (cant);
```

```
CODP    CANT_MEDIE
----------------------------
P3      100
P5      100
P4      325
P1      500
P2      500
```

Ca algoritm de analiză al funcției DECODE se poate observa că, pentru coloana DENP (primul argument) are loc căutarea valorilor "monitor 17 inch" şi "mouse A4 Tech", iar în cazul în care acestea sunt regăsite pe coloana denumirilor, prețurile lor sunt actualizate cu 1.1 şi, repsectiv, 1.2.

Pentru restul produselor care nu fac obiectul căutării, se trece implicit, ultimul argument, în cazul de față coloana PRET, sau se poate trece o expresie 'Pret_Nemodificat'

Fiind vorba de cumpărare, implicit se ia în calcul ca document de intrare NIR-ul (pentru aprovizionare), acest lucru necesitând o condiție suplimentară IE=1 (alături de cea care identifică din tabela Produse doar acele produse care au făcut obiectul tranzacției şi au la bază un document justificativ).

7) Să se afişeze doar acele produse care au cantitatea minimă vândută mai mare decât cantitatea minimă a produsului P3.

```
SQL> SELECT codp,
          MIN (cant) CANT_MINIMA
          FROM proddoc WHERE IE= -1
          GROUP BY codp
          HAVING MIN (cant) >
               (SELECT MIN (cant)
                    FROM proddoc
                    WHERE codp='P3');
```

```
CODP  CANT_MINIMA
----- --------------------------
P2              200
P4              500
```

Fiind vorba de vânzare se porneşte de la ideea că documentul justificativ aferent ieşirii din gestiune este avizul, ca atare, se va trece condiția IE=-1. Totodată, în această situație este vorba de o clauză select imbricată pentru a permite selecția doar a celor produse care respectă o condiție față de produsul P3. Ca şi în cazul anterior nu se va trece în clauza select atributul *cant* după pentru care se calculează funcțiile şi se face gruparea.

8) Să se afişeze cantitatea medie doar pentru produsele care apar mai mult de două ori în tabela Proddoc.

```
SQL> SELECT codp,
     AVG (cant) CANT_MEDIE
     FROM proddoc
     GROUP BY codp
     HAVING COUNT (*) > 2;
```

CODP	CANT_MEDIE
P1	300
P2	350
P3	100
P4	391.66667

9) Să se afişeze doar acele produse pentru care cantitatea este mai mare sau egală cu 200.

```
SQL> SELECT codp, MAX (cant) CANT_MAXIMA
     FROM proddoc
     HAVING MAX (cant) > = 200
     GROUP BY codp;
```

CODP	CANT_MAXIMA
P1	500
P2	500
P4	500

10) Să se afişeze doar acele produse pentru care cantitatea medie este mai mare sau egală cu 200.

```
SQL> SELECT codp, AVG (cant) MEDIE
     FROM proddoc
```

```
            GROUP BY codp
            HAVING AVG (cant) > 200;
```

CODP	MEDIE
P1	300
P2	350
P4	391.66667

11) Să se afişeze cantitatea medie pe tip de produs, pentru toate codurile de produs mai puțin P1.

```
SQL>  SELECT codp, AVG (cant) MEDIE_CANT
      FROM proddoc
      WHERE codp != 'P1'
      GROUP BY codp;
```

CODP	MEDIE_CANT
P2	350
P3	100
P4	391.66667
P5	100

12) Să se calculeze cantitatea medie pentru fiecare produs distinct, din tabela Proddoc.

```
SQL> SELECT codp,
     AVG (cant) MEDIE
     FROM proddoc
     GROUP BY codp;
```

CODP	MEDIE
P1	300
P2	350
P3	100
P4	391.66667
P5	100

13) Determinați prețul mediu pentru fiecare produs în afară de produsul 'Monitor 17inch' din tabela Produse.

```
SQL>  SELECT codp,
      AVG (pret) PRET_MEDIU
      FROM produse
      WHERE denp!= 'Monitor 17inch'
```

GROUP BY codp;

CODP	PRET_MEDIU
P2	1000000
P3	300000
P4	2700000
P5	100000

14) Afişaţi preţul minim pe produs.

```
SQL>  SELECT denp,
      MIN (pret) PRET_MINIM
      FROM produse
      GROUP BY denp;
```

DENP	PRET_MINIM
CD-RW ASUS 24x10x40x	1000000
CPU AMD Athlon 1.4GHz	2700000
Monitor 17inch	3500000
Mouse A4TECH	100000
Tastatura qwerty	300000

15) Să se afişeze toate produsele cu diferenţe cantitative în documente.

```
SQL>  SELECT a.codp
            FROM (
                  SELECT p.codp, SUM (cant) cant
                        FROM proddoc p,documente d
                        WHERE p.codd=d.codd
                              AND dend='FACT'
                        GROUP BY p.codp
                  ) a,
                  (SELECT p.codp, SUM(cant) cant
                        FROM proddoc p,documente d
                        WHERE p.codd=d.codd
                              AND dend<>'FACT'
                        GROUP BY p.codp
                  ) b
      WHERE  a.codp=b.codp
             AND a.cant-b.cant <> 0;
```

CODP
P4

Se identifică cu documentele FACTURA care se regăsesc atât în nomenclatorul de documente (tabela Documente) cât şi în nomenclatorul de documente “tranzacționate” (participante la tranzacțiile de produse, din tabela Proddoc). Apoi se identifică cu restul de documente (în afară de FACTURA) aflate atât în nomenclator, cât şi în tranzacții. în final se trec condițiile de identificare a produselor tranzacționate şi se stabilesc diferențele cantitative.

16) Să se afişeze denumirea, prețul şi valoarea totală a vânzărilor pentru fiecare produs, ținând cont de comisionul de 5%.

```
SQL>  SELECT denp, pret,
      SUM(pret*cant*1.05) TOTAL_VANZARI
      FROM produse, proddoc
      WHERE  produse.codp=proddoc.codp
             AND IE= -1
      GROUP BY denp,pret;
```

```
DENP                      PRET      TOTAL_VANZARI
--------------------------------------------------------------------------------
CD-RW ASUS 24x10x40x      1000000    210000000
CPU AMD Athlon 1.4GHz     2700000    1.418E+09
Monitor 17inch            3500000    1.470E+09
```

S-au identificat doar produsele pentru care IE=-1, respectiv au ieşit din gestiune (au fost vândute) fiind însoțite de documentul AVIZ (de expediție).

17) Să se afişeze valoarea maximă, valoarea medie, valoarea minimă şi valoarea totală pentru livrările (IE= -1) de produse efectuate.

```
SQL>  SELECT MAX (1.05*cant*pret) VZ_MAX,
      AVG (1.05*cant*pret) VZ_MED,
      MIN (1.05*cant*pret) VZ_MIN,
      SUM (1.05*cant*pret) VZ_TOTAL
      FROM produse, proddoc
      WHERE produse.codp=proddoc.codp
            AND IE= -1
      GROUP BY produse.codp;
```

```
VZ_MAX          VZ_MED VZ_MIN  VZ_TOTAL
--------- --------- --------- -------------------------------------
1.103E+09       735000000       367500000       1.470E+09
210000000       210000000       210000000       210000000
1.418E+09       1.418E+09       1.418E+09       1.418E+09
```

18) Să se calculeze şi afişeze profiturile rezultate din vânzari (IE= -1) cu comision de 5%

```
SQL>  SELECT p.codp,
      pret*0.95      PROFIT
      FROM produse p, proddoc pd
      WHERE p.codp=pd.codp
      AND IE= -1;
```

```
CODP    PROFIT
---- -------------------
P1      3325000
P2      950000
P1      3325000
P4      2565000
```

19) Să se afişeze tranzacțiile cu valoare mai mică decât cea mai mare valoare a unei tranzacții cu furnizorul 4

```
SQL>  SELECT codt, valoare
      FROM documente
      WHERE valoare < ANY
                  (
                  SELECT valoare
                  FROM documente d, tranzactii t
                  WHERE d.codt=t.codt
                        AND codf='4'
                  );
```

```
CODT    VALOARE
----- ---------------------
T2      1.255E+09
T3      550000000
T3      550000000
T5      570000000
T5      570000000
```

20) Afişați produsele care au cantitatea mai mare decât cea mai mică cantitate a produsului “P4” *(min(cant)P4=200).*

```
SQL>  SELECT codp, cant
      FROM proddoc
      WHERE
            Codp!= 'P4' AND cant > SOME
                              (SELECT DISTINCT cant
                              FROM proddoc
                              WHERE codp='P4')
      ORDER BY cant DESC;
```

CODP	CANT
P2	500
P1	500
P1	500
P2	500
P1	300
P1	300

Cea mai mică cantitate a produsului 'P4' este de 200 bucăți, astfel că, cererea principală întoarce toate produsele, cu excepția lui 'P4' (specificată explicit) care sunt într-o cantitatea mai mare decât minimul cantității produsului 'P4' specificat.

Astfel, condiția '> ANY' înseamnă *"mai mare ca minim"* iar '=ANY' este echivalent cu operatorul *IN*.

Când se foloseşte SOME/ANY, DISTINCT este frecvent utilizat pentru a împiedica să se selecteze liniile de mai multe ori.

21) Să se afişeze produsele care au cantitatea mai mare sau egală cu cea mai mare cantitate a produsului "P4" *(max(cant)P4=5200)*, inclusiv produsul 'P4'.

```
SQl>  SELECT codp, cant
      FROM proddoc
      WHERE cant > = SOME
                  (select MAX (cant)
                  FROM proddoc
                  WHERE codp='P4')
      ORDER BY cant DESC;
```

CODP	CANT
P2	500
P1	500
P1	500
P2	500
P4	500

P4 500
P4 500

22) Să se afişeze, pentru fiecare document în parte, ce procent reprezintă produsele din totalul de produse de pe document.

```
SQL> SELECT a.codd, a.codp, a.PROC/b.TOTAL*100  PROCENT
     FROM
          (SELECT codd, codp, cant PROC
               FROM proddoc
               GROUP BY codd, codp,cant) a,
          (SELECT codd, SUM (cant) TOTAL
               FROM proddoc
               GROUP BY codd) b
     WHERE a.codd=b.codd ;
```

CODD	CODP	PROCENT
10123	P1	50
10123	P2	50
10124	P3	14.285714
10124	P4	71.428571
10124	P5	14.285714
10125	P1	33.333333
10125	P2	66.666667
10126	P1	37.5
10126	P4	62.5
10127	P3	33.333333
10127	P4	66.666667
20123	P1	50
20123	P2	50
20124	P3	15.384615
20124	P4	69.230769
20124	P5	15.384615
20125	P3	33.333333
20125	P4	66.666667
30122	P1	33.333333
30122	P2	66.666667
30123	P1	37.5
30123	P4	62.5

23) Să se afişeze documentele având valorile totale cuprinse între 1.500.000.000 şi 6.500.000.000 sau cele ca sunt NIR-uri şi Facturi.

```
SQL> SELECT * from documente
     WHERE valoare BETWEEN 1500000000 AND 6500000000
     OR (dend='NIR' AND dend='FACT')
     ORDER BY data ASC;
```

CODD	DEND	DATA	CODT	VALOARE
20123	NIR	08-JAN-03	T1	2.250E+09
10123	FACT	08-JAN-03	T1	2.250E+09
30123	AVIZ	11-FEB-03	T4	2.400E+09
10126	FACT	11-FEB-03	T4	2.400E+09

24) Să se afişeze perioada de timp, în săptămâni rămase până la expirarea fiecărui produs. Săptămânile (cu perioadele interimare rezultate) se vor rotunji (prin funcțiile ROUNDsau TRUNC) la valorile întregi.

SQL> SELECT termen,
ROUND ((termen-sysdate)/7) SAPT_GARANTIE
FROM produse;

TERMEN	SAPT_GARANTIE
01-AUG-06	126
01-AUG-05	74
01-JUN-04	13
01-DEC-04	39
01-JUN-04	13

25) Să se afişeze denumirea şi valoarea documentelor împreună cu data incheierii lor, doar pentru tranzacțiile încheiate în luna februarie 2005.

SQL> SELECT dend, valoare, data DATA_INCHEIERII
FROM documente
WHERE TO_CHAR (data, 'MM/YY') = '02/05';

DEND	VALOARE	DATA_INCHEIERII
FACT	570000000	27-FEB-05
NIR	570000000	29-FEB-05

26) Să se creeze un index nou pe atributul denumire produs (Denp) din tabela Produse.

SQL> CREATE INDEX prod_idx
ON produse (denp);

Index created.

27) Să se afişeze indecşii creaţi pentru tabela Produse şi dacă asigură unicitatea.

```
SQl>  SELECT
      IC.index_name,  IC.column_name,
      IC.column_position COL_POZ,
      IX.uniqueness
      FROM user_indexes IX, user_ind_columns IC
      WHERE IC.index_name=IX.index_name
      AND IC.table_name= 'PRODUSE ';
No rows selected.
```

28) Să se şteargă indexul creat anterior.

```
SQl>  DROP INDEX prod_idx;
Index dropped.
```

29) Să se afişeze restricţiile definite pentru tabela Tranzacţii.

```
SQL>  SELECT CONSTRAINT_TYPE      TIP_RESTR,
      CONSTRAINT_NAME             NUME_RESTR,
      STATUS                      STAREA_RESTR
      FROM USER_CONSTRAINTS
      WHERE TABLE_NAME= 'TRANZACTII';
```

TIP_RESTR	NUME_RESTR	STAREA_RESTR
C	NN_DENT	ENABLED
C	CK_DENT	ENABLED
P	PK_CODT	ENABLED
R	FK_CODF	ENABLED
R	FK_CODC	ENABLED

30) Să se afişeze numele tabelelor create în schema proprie de obiecte

```
SQL>  SELECT table_name FROM USER_TABLES ;
```

TABLE_NAME
BONUS
CLIENTI
DEPT
DOCUMENTE
EMP
FURNIZORI
PRODDOC
PRODUSE

SALGRADE
TRANZACTII
10 rows selected.

31) Să se afişeze tipurile de obiecte create în schema proprie de obiecte

SQL> SELECT DISTINCT OBJECT_TYPE
FROM USER_OBJECTS;

OBJECT_TYPE

INDEX
SEQUENCE
TABLE

32) Să se redenumească tabela Clienţi în tabela "Clienţi_Redenumiţi".

SQL> ALTER TABLE clienti RENAME TO clienti_redenumiti;
Table altered.
SQL> SELECT * FROM Clienti_Redenumiti;

CODC	DENC	ADR	LOC	CONT	BANCA
1	CONN	PIPERA 135	BUCURESTI	A1234567890	BRD
5	Flanx	Dorobantilor 130	Cluj	C1231231234	BCR

33) Să se şteargă tabela Clienţi_Redenumiţi şi să se elibereze spaţiul ocupat de aceasta.
SQL> TRUNCATE TABLE clienti_redenumiti;

34) Să se creeze un sinonim public pentru tabela Produse din schema de obiecte Student.

SQL> CREATE PUBLIC SYNONYM prod FOR student.produse;

35) Să se creeze utilizatorul AGENT001 cu parola Agent.

SQL> CREATE USER AGENT001
IDENTIFIED BY agent;

36) Să se creeze o serie de drepturi la nivel de sistem pentru utilizatorul AGENT001.

SQL> GRANT CREATE TABLE,
CREATE SEQUENCE,

CREATE VIEW
TO AGENT001;

37) Să se modifice parola utilizatorului AGENT001 cu "noua_parola"

SQL> ALTER USER AGENT001
IDENTIFIED BY noua_parola;

38) Să se creeze rolul AgVanz cu drepturile RESOURCE si CONNECT la nivel de sistem.

SQL> CREATE ROLE agvanz;
SQL> SET ROLE AgvVanz;
SQL> GRANT RESOURCE, CONNECT TO AgVanz;

39) Să se ataşeze rolul AgVanz utilizatorului AGENT001.

SQL> GRANT AgVanz TO AGENT001;

40) Să se anuleze drepturile primite de utilizatorul AGENT001 pe tabela Documente.

SQL> REVOKE ALL ON documente FROM AGENT001;

41) Să se creeze tabela partiționată Vânzari (codt, data, suma) cu partiții pentru vânzarile din ultimele 3 luni.

SQL> CREATE TABLE
vanzari (codt varchar2 (5), data date, suma number (11))
STORAGE (INITIAL 100K NEXT 50K) LOGGING
PARTITION BY RANGE(data)
(PARTITION LUNA03 VALUES LESS THAN (4)
TABLESPACE T0,
PARTITION LUNA02 VALUES LESS THAN (3)
TABLESPACE T1,
PARTITION LUNA01 VALUES LESS THAN (2)
TABLESPACE T2);

42) Să se adauge noi tupluri din tabela Tranzacții în partiția LUNA02 din tabela Vânzări.

```
SQL> INSERT INTO vanzari
     PARTITION (LUNA02)
     SELECT T.codt,
     TO_CHAR (dataora,'MM-DD-YYYY'), valoare
     FROM tranzactii T, documente D
     WHERE T.codt=D.codt
     AND MONTHS_BETWEEN (data, sysdate)=2;
```

43) Să se creeze un raport care să afişeze informaţiile despre tranzacţiile încheiate în ultimul an, documentele şi produsele aferente şi un total general.

```
SQL> SET PAGESIZE 200
SQL> SET LINESIZE 100
SQL> SET FEEDBACK OFF
SQL> SET ECHO OFF
SQL> SET VERIFY OFF
SQL> COLUMN CODT FORMAT a5 HEADING 'Nr'
SQL> COLUMN DENT FORMAT a1 HEADING 'Tip'
SQL> COLUMN DATAORA FORMAT a10 HEADING 'Data'
SQL> COLUMN DEND FORMAT a4 HEADING 'Doc'
SQL> COLUMN CODD FORMAT 99999 HEADING 'Nr Doc'
SQL> COLUMN CODF FORMAT a5 HEADING 'Fz'
SQL> COLUMN CODC FORMAT a5 HEADING 'Cl'
SQL> COLUMN VALOARE FORMAT 99999999999
     HEADING 'Valoare Tranzactie'
SQL> COLUMN CODP FORMAT a5 HEADING 'Produs'
     COLUMN CANT FORMAT 99999 HEADING 'Cantitate'
SQL> SELECT T.codt, dent, dataora, dend,
     D.codd, codf, codc, valoare, codp, cant
     FROM tranzactii T,documente D, proddoc P
     WHERE       T.codt=D.codt
                 AND D.codd=P.codd
     ORDER BY  T.codt, Dcodd, codp
```

Nr	T	Data	Doc	Nr Doc	Fz	Cl	Valoare	Produs	Cantitate
T1	R	08-JAN-04	FACT	10123	3	1	2250000000	P1	500
T1	R	08-JAN-04	FACT	10123	3	1	2250000000	P2	500
T1	R	08-JAN-04	NIR	20123	3	1	2250000000	P1	500
T1	R	08-JAN-04	NIR	20123	3	1	2250000000	P2	500
T2	R	10-NOV-04	FACT	10124	4	1	1390000000	P3	100
T2	R	10-NOV-04	FACT	10124	4	1	1390000000	P4	500
T2	R	10-NOV-04	FACT	10124	4	1	1390000000	P5	100

ANEXE

ANEXA 1

CUVINTE REZERVATE

LIST ROWID
LOCK ROWNUM
ACCES DISTINcT LONG ROWS
ADD DOES MAJCEXTENTS RUN
ALL DROP MINUS SELECT
ALTER MODE SESSION
AND ELSE MODIFY SET
ANY ERASE MOVE SHARE
APPEND EVALUATE NEW SIZE
AS EXCLUSIVE NOAUDIT SMALLINT
ASC EXISTS NOCOMPRESS SPACE
ASSERT FILE NOLIST START
ASSIGN FLOAT NOSYSSORT SUCCESSFUL
AUDIT FOR NOT SYNONYM
BETWEEN FORMAT NOWAIT SYSDATE
BY FROM NULL SYSSORT
CHAR GRANT NUMBER TABLE
CHECK GRAPHIC OF TEMPORARY
CLUSTER GROUP OFFLINE THEN
COLUMN HAVING OLD TO
COMMENT IDENTIFIED ON TRIGGER
COMPRESS IF ONLINE UID
CONNECT IMAGE OPTIMIZE UNION
CONTAIN IMMEDIATE OPTION UNIQUE
CONTAINS IN OR UPDATE.
CRASH INCREMENT ORDER USER
CREATE INDEX PAGE USING
CURRENT INDEXED PARTITION VALIDATE
DATAPAGES INDEXPAGES PCTFREE VALUES
DATE INITIAL PRIOR VARCHAR
DBA INSERT PRIVILEGES VARGRAPHIC
DBLINK INTEGER PUBLIC VIEW
DECIMAL INTERSECT RAW WHENEVER
DEFAULT INTO RENAME WHERE
DEFINITION IS REPLACE WITH
DELETE LEVEL REPORT
DESC LIKE RESOURCE
REVOKE

ANEXA 2

*OPERATORI UTILIZATI IN LIMBA JUL SQL*PLUS*

*A. Sintaxa operatorilor SQL*PLUS*

Operator	**Functie**
&	Specifică înlocuirile lexicale într-un fişier de comenzi executat cu START. Opțiunile sunt înlocuite prin &s: prima prin &1, a doua prin &2 etc.
&,&&	Indică o variabilă utilizator într-o comandă SQL*PLUS. Se cere o valoare de fiecare dată când variabila & este găsită şi se cere o valoare când se găsete prima dată variabila &&. Incheie o variabilă de substituție urmată de un caracter ce ar putea fi parte a numelui variabilei.

B. Sintaxa operatorilor SQL

Operator	**Functie**
(…)	Include o subcerere conținută ântr-o altă comandă
’…’	Delimitează o constantă de tip caracter sau dată calendaristică. Un apostrof într-o constantă de tip caracter se preprezintă prin două apostrofuri.
“…”	Marchează un nume de coloană sau sinonim care conține caractere speciale. Marchează literalii într-un format de tip dată calendaristică.
@	Precede un nume de legătură la o bază de date, într-o clauză FROM.

C. Operatori aritmetici SQL

Operator	**Functie**
+, -	Operatori unari + şi – (valori pozitive, respectiv valori negative)
*, /	Operatori de înmulțire şi împărțire
+ -	Operatori de adunare şi scădere
||	Concatenare de şiruri de caractere

D. Operatori logici

Operator	Functie
<, >, =, >=, <=, !=, <>	Operatori de comparatie
NOT	Funcția logică „NU”
AND	Funcția logică „ŞI”
OR	Funcția logică „SAU”
[NOT] IN(listă)	Egal cu oricare valoare din lista de valori
ANY	Indică o valoare oarecare dintr-o mulțime
ALL	Indică toate valorile unei mulțimi
[NOT] BETWEEN X AND Y	Valoarea unei variabile [nu] se găseşte în intervalul [x,y]
EXISTS	Condiția este adevărată dacă o subcerere returnează cel puțin un rând
[NOT] LIKE	Compară valoarea unui câmp cu un şir de caractere. Dacă în şirul de caractere urmează % se compară cu orice secventă de caractere; _ - se compară cu orice caracter;
IS [NOT] NULL	Operatorul specifică dacă valoarea unei variabile este sau nu nulă

E. Operatori utilizați în expresiile de cereri

Operator	Functie
UNION	Combină mai multe cereri, returnând reunirea liniilor selectate de cererile individuale
INTERSECT	Combină mai multe cereri, returnând intersecția liniilor selectate de cererile individuale
MINUS	Combină mai multe cereri, returnând diferența liniilor selectate de două cereri (rândurile distincte selectate de prima cerere şi neselectate de a doua cerere)

F. Alți operatori SQL

Operator	Functie
(+)	Indică faptul că o coloană ce îl precede este coloană de joncțiune externă (OUTER JOIN COLUMN)
PRIOR	Definşte relațiile părinte-fiu între nodurile unei cereri structurate arborescent. Dacă operatorul PRIOR precede o expresie din stânga unei egalități, se face o selecție descendentă. Dacă operatorul PRIOR precede o expresie din dreapta unei egalități se va face o selecție ascendentă.

ANEXA 3

*PSEUDOCOLOANE UTILIZATE IN LIMBA JUL SQL*PLUS*

Număr coloană	**Valoare returnată**
LEVEL	1 pentru rădăcină, 2 pentru subdirector (copil) al rădăcinii etc. Se utilizează în comanda SELECT şi cluza CONNECT BY
NULL	Returnează o valoare nulă. Nu poate fi folosită în expresiile logice.
ROWID	Numărul de identificare al rândului. Un identificator de rând este de tipul ROWID şi nu de tipul număr sau caracter.
ROWNUM	Numărul de ordine al rândului selectat din tabelă (numărând de la 1)
SYSDATE	Data calendaristică şi timpul curent
UID	Identificator de utilizator, este un număr unic pentru fiecare utilizator
USER	Returnează numele utilizatorului curent